A Misuse Pattern for Web Browsers: Interception of traffic

Paulina Silva
Departamento de Informática
Universidad Técnica Federico
Santa María
Valparaíso, Chile
pasilva@alumnos.inf.utfsm.cl

Raúl Monge
Departamento de Informática
Universidad Técnica Federico
Santa María
Valparaíso, Chile
rmonge@inf.utfsm.cl

Eduardo B. Fernandez
Department of Computer &
Electrical Engineering and
Computer Science
Florida Atlantic University
Boca Raton, Fl, USA
fernande@fau.edu

ABSTRACT

Currently, most software development is focused in creating systems connected to the Internet, which allows to add functionality within a system and facilities to their *Stakeholders*. This leads to depend on a *web client*, such as *Web Browser*, which allows access to services, data or operations that the system delivers. However, the Internet influences the attack surface of the system, and unfortunately many stakeholders and developers are not aware of the risks to which they are exposed. The lack of security education among software developers and the scarce and scattered documentation for browsers (and standardization) could become a big problem in large architectural developments that depend on browsers to perform their services. We are studying some security attacks in the web browser by describing them in the form of misuse patterns. A misuse pattern describes how an information misuse is performed from the point of view of the attacker. It defines the environment where the attack is performed, how the attack is performed, countermeasures to stop it, and how to find forensic information to trace the attack once it happens. We are building a catalog of misuse patterns and we present here one we call Interception of traffic in the Web Browser. A catalog of misuse patterns will help designers to evaluate their designs for possible threats.

Keywords

Browser, Web Client, Misuse Pattern, Security, Man-in-the-Browser

1. INTRODUCTION

The current scenario of attacks in the browser has changed considerably, if compared to those browsers in the 90s. Every day browsers are more robust and difficult to exploit; therefore, the same attack types, such as drive-by downloads or code-based execution that could subvert a system, are less common every time. A new form of attack has emerged and is fairly easy to achieve, because it is based on deceiving the user to perform what the attacker wants. Once the user is tricked, the attacker can achieve total control over the browser or the host, without having to crack the system [1, 2] that hosts the browser. The development of critical systems that interact daily with different users on the network should focus on these attacks because they threaten the confidentiality, integrity and availability of the user's data (personal) as well as the Stakeholders involved.

The new type of attacks described above are called "social engineering attacks", in [3] they are defined as: The act of manipulating someone to perform actions that are not part of the best interests of the victim (person, organization, stakeholder, etc). An attack of this kind can take many forms, there is the possibility of a physical or digital encounter with the victim. Based on social engineering, this attack is one that takes advantage of human behavior and trust of the victim. In the context of web browser, the deceived user is the first and last line of defense against such attacks, the abuse of trust of the user may open the doors of the browser's host, causing damage to both the user and the external systems with which it interacts.

According to studies [4, 2, 5], the browser is the first line of defense against multiple Web threats. However, this is affected by the lack of education of users who use browsers and the constant evolution of threats [4]. This is why many browser manufacturers have created defense mechanisms such as those shown in [6], that act when the user requests a page, including black or white lists, reputation systems [1] with warning alerts, among others, so the user can at least avoid the page or choose to enter the malicious site anyway (but no granting access to the page without knowing of the threat).

In this work, a Misuse Pattern is presented as a first step in to the process of creating a catalog of misuses for the Web Browser. A catalog of misuse patterns will help designers to evaluate their designs for possible attacks.

AsianPLoP 2016 February 24-26, Taiwan.
ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

2. MISUSE PATTERN: INTERCEPTION OF TRAFFIC IN THE WEB BROWSER

Intent

An attacker could modify or spy the traffic when the web Browser User receives the response from the Provider to the Host; by doing so, the browser could interpret the information in a different way than if it had received the original traffic.

Context

A web browser fetches resources (web pages, services, etc.) from a Provider to satisfy the access of a Browser User. The Provider is generally a Web App or Web server, that allows input and output of data to other applications, and usually they are built using HTML, Javascript and CCS. A Provider, depending on its type, can receive many requests for resources from various hosts. Depending on the type of request, they may or may not be accepted. For those accepted, the Provider generates a response to the Host, which may go back (or not) to the Browser Client that generated the request.

Problem

An attacker tries to take advantage of any input the Browser User requests to affect the system. Users may be tricked by social engineering attacks, or lead into downloading and executing a binary in the browser to affect the Renderer that may have unprotected vulnerabilities. Therefore it is possible to give the attacker a chance to hide in the middle of the communication between the Host and the Provider, resulting in the interception of content that may affect the Web Content Renderer or the Browser Client. Depending on the type of attacker, it is possible that it may even affect the host where the browser is.

The solution is affected by the following forces:

- Misuse: Perform some destruction and/or other misuses (confidentiality and integrity).
- Stealthy/Untraceability: Try to hide its structure to make harder its detection and removal. Since it can compromises the host, it would be better if no one knows who is the responsible one.
- Collateral damage: In addition to specific misuses, the attack may require difficult operations for stopping or disrupting browsers activities; web browser's basic knowledge is needed.
- Activation: This can be done by enticing offers which may tempt users to open email attachments or download procedures (social engineering).

The attack could take advantage of the following vulnerabilities:

- The The Same Origin Policy (SOP) is defined by the **origin**, and is used to separate different resources by its domain, scheme and port (i.e: somedomain.com, http and 80). It is the minimun security mechanism a browser has while requesting cross-origin resources, and divides the different kind of contents so they can not interfere with each other. The SOP which every browser complies with, differs in every web browser manufacturer [7, 8, 9, 10, 11]; for this reason, attackers take advantage to make malicious cross-origin requests.
- Attacker can take advantage of the flexibility the SOP has, because the **origin** is not enough as an isolation mechanism between the different resources [12, 13, 14, 15]. Different levels of isolation can give better results, but affecting the performance of the browser [16, 17]
- Anyone can create a software component like an extension or plugin for some type of web browser and pass it off as something harmless, consequently a user will not notice the threat and will install it.
- It is possible to affect the Browser Client, and in consequence the Host, without having to find a vulnerability in the system or browser. With social engineering methods it is possible to trick the user/victim into doing things that are not part of the best interests of the victim, because the Browser User is the weakest link in the system.
- The architecture to extend the browser functionality through extensions, plugins and other, depends on the manufacturer, and probably it has a large attack surface.
- No use or improper use of the Sandbox (No access control).
- Many browser manufacturers do not have robust defense mechanisms that allow an effective identification of malicious resources.

The attack can be facilitated by:

- There are many tools for social engineering attacks, that tricks the Browser User into accepting the installation of extensions or malicious plugins more easily.
- Specially crafted scripts can be used to exploit the interpreter within the Renderer of the web browser (represented here as the Web Content Renderer). Often it is also possible to use the same scripting language elements to pass through certain security barriers provided by the SOP because the language is based on prototypes (ECMAscript). A prototype language is a style of object-oriented programming in which behaviour reuse or inheritance is performed via a process of cloning existing objects that serve as prototypes. Since the prototype is cloned for creating other objects, this could be uses to find new vulnerabilities within the Renderer of browsers.
- Encryption methods can do nothing against an attack that intercepts traffic before sending or after receiving a requested service/resource.
- There are browsers that still use a monolitic architecture (monoprocess). Meaning that the browser is using the users permission to access directly the host resources.

Solution

An attacker can take advantage of the Browser User by letting a social engineering attack do its job, by surfing every website without distrust. If the Browser User is not careful, while surfing in the Internet, it could lead into downloading and installing malicious binaries in the system without notice. Also, a single phishing e-mail wishing for the user to click in a URL address could lead into installing a binaries, extensions or scripts in the Host. If a social engineering attack is successful the attacker can take its time, because whatever is installed in the Host, the attacker has installed it with the user's permission. Therefore every action done in the Host or Browser Client, will be identified by the Host as an action done by the Browser User, a user of the Host. If the attacker can obtain physical access to the Host running the Browser Client, he or she can install without the real user's consent an extension, script or binary in the Host, that could lead to a misuse. Also, is not a necessity for the attacker to install malicious payload/binaries in the Browser, she or he could use "benign-but-buggy" extensions or scripts to surpass the Sandbox's secure mechanism and misuse the Host or Browser Client.

Structure

The structure of the solution used is the same as in the Browser Infrastructure Pattern [18]. In Figure 1 the **Browser Client** is an entity that represents the main process of a Web Browser, which is constantly communicating with the host of the browser. A **Host** houses and interacts with the **Browser Client**. The **Host** uses the **Graphic Processing Unit (GPU)** package to display elements to the **Browser User**. A user who makes a request to a Internet resources using a Web Browser, will be called **Browser User**. At the same time, a **Provider** has a **Web Server** responsible for receiving external requests from different **Hosts**. According to the request, a **Provider** will send the **Service** (or resource) the **Browser User** needs. Most Browsers use a central component to do operations that need to affect the Host of the Browser, a **Browser Client**. Figure 1 shows the Class diagram for the Browser Infrastructure Pattern. For each new resource a **Browser Client** requests a created or reused **Web Content Renderer** instance. A **Plugin** or **Extension** are element that extends the Browser's functionalities, the difference between the two of them is that the latter is used exclusively within the browser while a **Plugin** can be used without it. A **Sandbox** is a controlled execution domain created for a single **Web Content Renderer** instance, that performs the access control of each communication between the **Web Content Renderer** and the **Browser Client**, as a broker with fine control over the messages (using IPC/IPDL/COM); the same applies to a **Plugin**, an **Extension** and a **GPU**. Depending on the manufacturer, a **Plugin** could not be Sandboxed. If there is a need for **GPU**, from **Extensions** or **Plugins**, they need to communicate directly with the Sandbox which instead will ask for graphical processing. The Attacker class is a person using some unit that could undertake a risky action against the integrity and confidentiality of the browser, the user, Host and Provider (Figure 1). The attacker is able to intercept both the response sent to the Browser Client from the Provider using the Host as a receiver or by having the browser make changes to the input, using scripts or other resources.

Dynamics

In Figure 2 a series of required steps is shown, for one of the many misuses that can be made for the use case **Make Request**. The attacker is located between the Browser Client and the Host, intercepting the original request or response and modifying the traffic to its taste; usually an attack basd on this misuse is called Man-in-the-Browser (MITB) [15, 19, 20, 21]. This could also happen when the Browser User has allowed the installation of plugins, extensions or external programs in the Host and Browser Client.

Summary

The attacker intercepts the traffic between the Host and Browser Client. This action could lead to different kinds of misuses or new attacks depending on the attacker's intentions.

Actor

Attacker

Precondition

The Browser Client runs with the Browser User's permissions (a user in the Host), while interacts with limited privilege processes.

Description

1. An attacker uses a social engineering technique or vulnerability in the system, and creates an entity between the Browser Client and Provider and their communication channel, normally a plugin, binary or extension. Also, an attacker could be using a "benign-but-buggy" extension or script to the same purpose.

2. A Browser User requires a browser to access a URL for some resource in a Provider, this is done by using a Browser Client already instanced in the Host. With a Sandbox there is an instance of Web Content Renderer pattern.

3. The Sandbox requires the Host resources to obtain what is behind the URL. A request is made from the Sandbox to the Browser Client through a communication channel such as IPC, IPDL or COM (depending on the *Browser* used) using a limited API to communicate to a process of greater privilege.

4. The Browser Client receives the request, and verifies through its policy engine if the Sandbox action is allowed.

5. At the time the Browser Client should be ready to proceed as usually, a plugin, an extension or a program in the Host will intercept the message before the system call is done, as the Browser Client has been tapped to perform that action.

6. The attacker then receives all traffic from the Browser Client, which could be modified or listened.

Postconditions

The victim will be fully compromised and it probably will not be possible to detect the modification of a message, it is also possible that the log of the Host will be affected.

Known Uses

The browser is a software that has different implementations, so the number of attack vectors are significant. Some of these are:

- Zeus [22] is a Trojan horse malware that runs on Microsoft Windows and can be used to carry out many malicious and criminal tasks, it is often used to steal banking information by man-in-the-browser. First identified in July 2007 when it was used to steal information from the United States Department of Transportation.

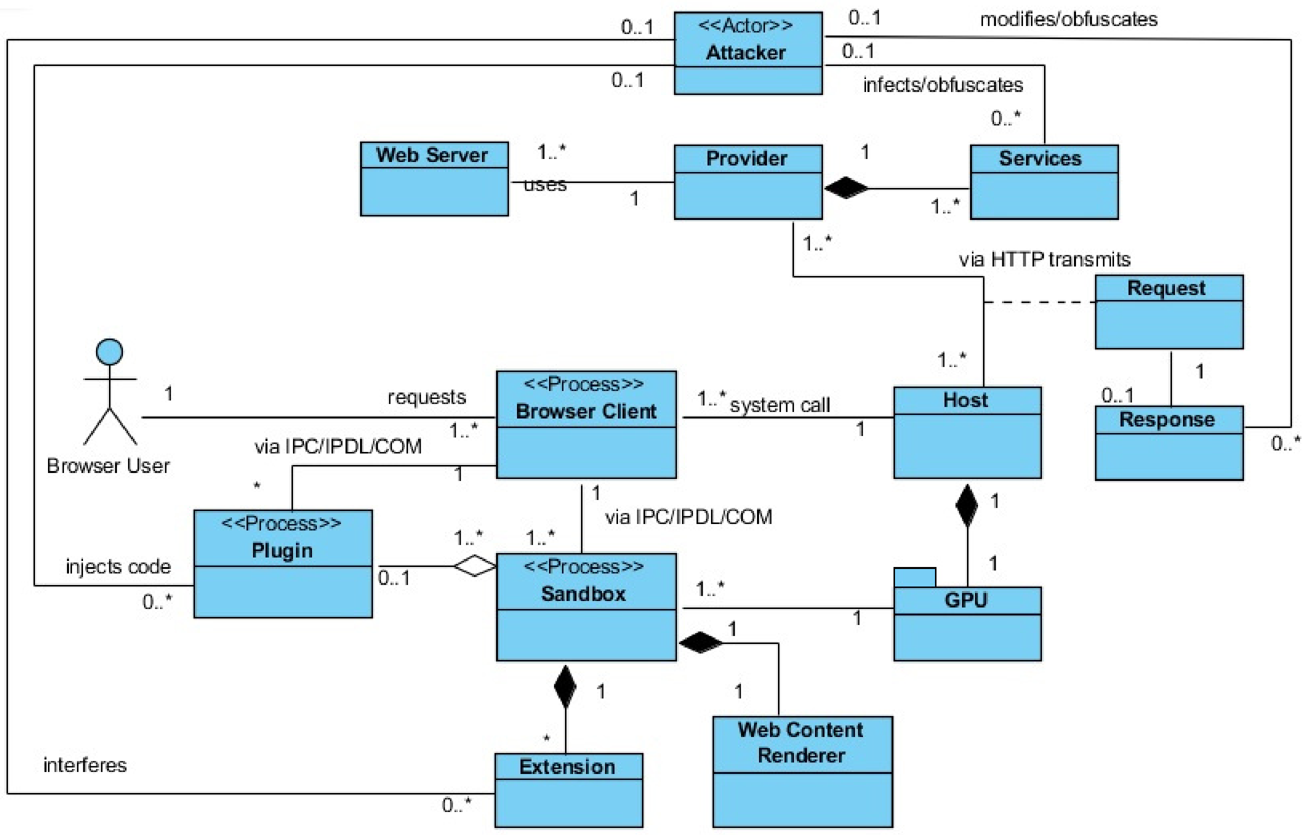
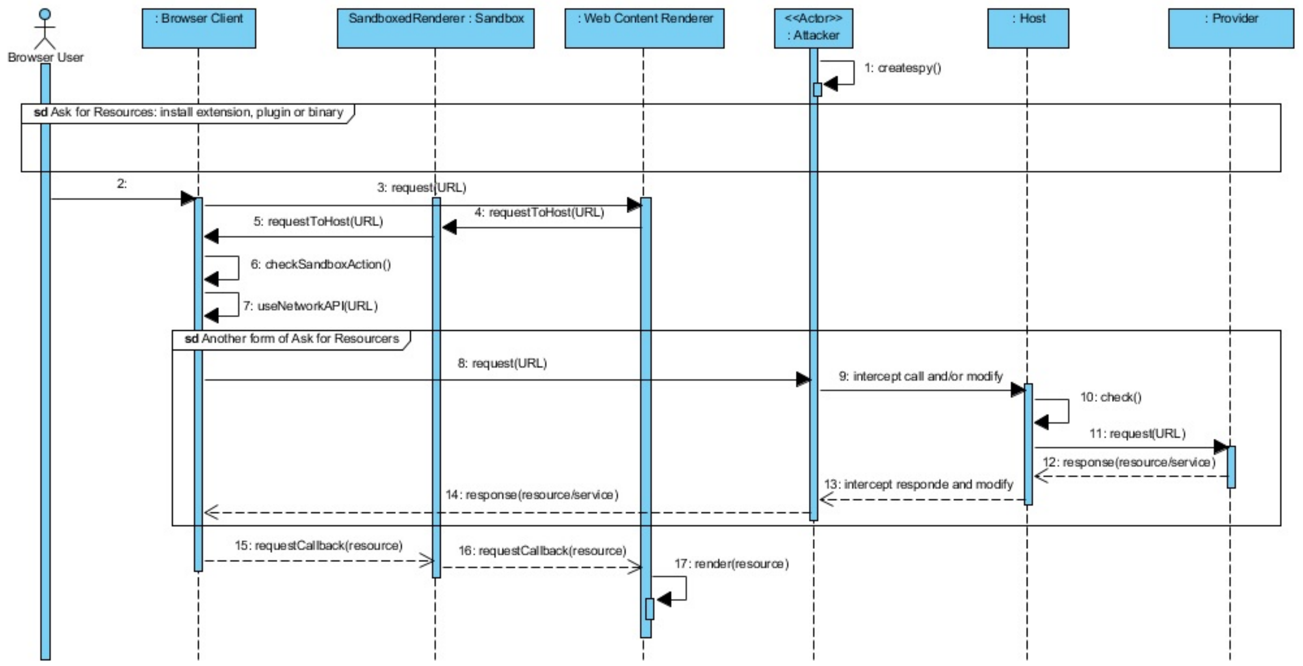


Figure 1: Class Diagram for the Misuse Pattern.



Browser User
: Browser Client
SandboxedRenderer : Sandbox
: Web Content Renderer
<<Actor>>
: Attacker
: Host
: Provider
1: createspy()
sd Ask for Resources: install extension, plugin or binary
2:
3: request(URL)
4: requestToHost(URL)
5: requestToHost(URL)
6: checkSandboxAction()
7: useNetworkAPI(URL)
sd Another form of Ask for Resourcers
8: request(URL)
9: intercept call and/or modify
10: check()
11: request(URL)
12: response(resource/service)
13: intercept responde and modify
14: response(resource/service)
15: requestCallback(resource)
16: requestCallback(resource)
17: render(resource)

Figure 2: Sequence Diagram for the misuse: Interception of traffic in the Web Browser.

- SpyEye [23] is known as Zeus's successor, it is sophisticated botnet creation kit that has been implicated in a number of costly online banking thefts against businesses and consumers. One way people get infected is by visiting a website that has been tampered with by hackers. The site will contain a 1x1 pixel that pulls JavaScript from a different server and begins testing to see if the victim's computer has unpatched software.

- An extension based on the Google Chrome architecture or the Firefox WebExtension API could intercept the data before it reaches the Browser Client or Host [11]. It could also be possible that a vulnerability in the extension or plugin is used by an attacker and takes advantage of its functionality to attack [15, 19]. Since the plugin, extension or process are elements the Host trust, it is possible that the attack is undetectable and the encryption methods do not serve as a mitigation measure.

- This type of attack can be used as base for more advanced attacks. For example, the browser could have *cross-origin javascript capability leak* vulnerabilities, when different security models such as the one used by Javascript and the DOM interfere with each other. As a consequence, a *cross-origin* request can perform even when the SOP was supposed to stop such attack [13]. In 2009 a security bulletin regarding this type of vulnerability was issued [24], where the getSVGDocument method was lacking an access check and allowed malicious a web site operator use the leaked capability to inject JavaScript into a target web site hosting an SVG document, bypassing the same-origin policy.

Consequences

The misuse has the following consequences for the attacker:

- Misuses: they may be different, we highlight vandalism, impersonate another person or monetary gain. While the attacker may be between the host and the traffic that is sent to the Provider, confidentiality and integrity of the data is completely lost even if the browser is communicating with a Provider through a secure channel. User privacy can no longer be assured.

- Stealthy/Untraceability: Since the attacker has managed to come between the system calls, that are made to the host, to send data to the Provider, the Host will not recognize or log the anomaly. Calls made to Host are perfectly legal and nothing out of the ordinary, so it will not be seen as something suspicious.

- Collateral damage: The attacker could perform actions that affect the integrity of the Host. The cost of fixing whatever the attack made on the host could have financial and organizational consequences.

- Activation: This can be done by enticing offers which may tempt users to open email attachments or download procedures (social engineering).

Possible sources of failure:

- If the Browser User is able to avoid or ignore the social engineering attack carried out at the beginning, this misuse can be avoided.

- Also, this should consider that the user does not encounter pages with malicious content, which may affect other parts of a browser, but that would cause the same effect as the misuse presented here.

Countermeasures

To prevent this kind of misuse we recommend taking the following preventive measures:

- Reputation services such as SmartScreen [25] from Internet Explorer and Download Application [1] from Google Chrome, can help identify pages, web content or resources that could contain malware when is installed as plugins, extensions or process in the Host of the User Browser.

- Providing education about the dangers while surfing the Internet and clarifying the users that they are the last line of defense against such attacks.

- Whitelist and Blacklist are installed in the browser as a preventive measure. They help avoid malicious pages or known malware while the user is browsing, they also are updated in a hourly-basis.

- Browsers like Google Chrome and Internet Explorer offer Sandboxing [26, 14]. This defense mechanism limits the actions of the attacker which may affect the integrity of the system.

Forensic Evidence

Where is it possible to find evidence? Depending on what is desired by the attacker, actions may differ. However the internal log of the browser could help in the audit of the system. This works until an attacker finds a vulnerability in the Sandbox or other component of the browser, in which case he can completely erase its tracks. Also, malware detection such as antivirus systems could help identifying tracks of misuses.

Related Patterns

- The Browser Infrastructure pattern [18].

- This pattern, has a class called Browser Client that acts as a Reference Monitor [27].

- Blacklist and Whitelist patterns [27].

3. CONCLUSIONS AND FUTURE WORK

We have presented a web browser attack as a misuse pattern that systematically describes how a misuse is performed. The aim is to understand and visualize the misuses of the browser that communicates with other systems, mainly to teach developers who have little (or none) security expertise. Through the list of threats shown in our previous work, it is possible to detect or infer misuse activities that may appear in one or more misuse cases.

With this misuse pattern we intent to initiate a catalog. This would help to condense the knowledge obtained using patterns so they can be used as guidelines to communicate relevant concepts, as well as evaluate the existent relationship between the browser and a developed system, to see what kind of interactions they have.

Future work to do is finishing the Reference Architecture for *Web Browsers*. Other patterns related to Browser Infrastructure pattern will be obtained in order to complete the Reference Architecture, such as the Web Content Renderer and Browser Kernel pattern.

We plan to build more Misuse Patterns for the Browser Infrastructure Pattern, to continue the study of the possible threats in the *Browser*, as a way to educate Developers and Stakeholders. At the same time, these patterns will allow the construction of the Security Reference Architecture for the browser. In the same line, in addition to finding potential threats existing in the system, we need to find countermeasures or security defenses to prevent or foresee such threats through security patterns on the reference architecture built. An example of the type of work to be carried out can be seen in [28].

4. REFERENCES

[1] M. Rajab, L. Ballard, and N. Lutz, "CAMP: Content-agnostic malware protection," *Proceedings of Annual . . .* , 2013. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.295.6192\&rep=rep1\&type=pdf

[2] NSS Labs, "Evolutions In Browser Security," no. October, pp. 1–20, 2013.

[3] J. Talamantes, "The social engineer's playbook." [Online]. Available: http://www.thesocialengineersplaybook.com/

[4] R. Abrams, J. Pathak, and O. Barrera, "Browser security comparative analysis: Phishing protection," 2013.

[5] ——, "Browser Security Comparative Analysis: Socially Engineered Malware Blocking," 2014.

[6] J. Drake, P. Mehta, C. Miller, S. Moyer, R. Smith, C. Valasek, and A. Q. Approach, "Browser Security Comparison," *Accuvant Labs*, 2011.

[7] W3C, "Same Origin Policy," W3C, Web page, 2010. [Online]. Available: https://www.w3.org/Security/wiki/Same_Origin_Policy

[8] C. Reis and S. D. Gribble, "Isolating web programs in modern browser architectures," *Proceedings of the fourth ACM european conference on Computer systems EuroSys 09*, vol. 25, no. 1, p. 219, 2009. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1519065.1519090

[9] C. Jackson and A. Barth, "Beware of finer-grained origins," *Web 2.0 Security and Privacy*, 2008. [Online]. Available: http://seclab.stanford.edu/websec/origins/fgo.pdf

[10] M. Crowley, *Pro Internet Explorer 8 & 9 Development: Developing Powerful Applications for The Next Generation of IE*, 1st ed. Berkely, CA, USA: Apress, 2010.

[11] S. D. Paola and G. Fedon, "Subverting Ajax," *23rd Chaos Communication Congress*, no. December, 2006. [Online]. Available: http://events.ccc.de/congress/2006/Fahrplan/attachments/1158-Subverting_Ajax.pdf

[12] M. Silic, J. Krolo, and G. Delac, "Security vulnerabilities in modern web browser architecture," *MIPRO, 2010 Proceedings of the 33rd International Convention*, 2010.

[13] A. Barth, J. Weinberger, and D. Song, "Cross-Origin JavaScript Capability Leaks : Detection , Exploitation , and Defense," *Opera*, vol. 147, pp. 187–198, 2009.

[14] M. V. Yason, "Diving into IE 10's Enhanced Protected Mode Sandbox."

[15] L. Liu, X. Zhang, G. Yan, and S. Chen, "Chrome extensions: Threat analysis and countermeasures," *. . . of the Network and Distributed Systems . . .* , 2012. [Online]. Available: https://www.cs.gmu.edu/~sqchen/publications/NDSS-2012.pdf

[16] A. Barth, C. Jackson, C. Reis, T. Team *et al.*, "The security architecture of the chromium browser," 2008.

[17] "Site Isolation - The Chromium Projects." [Online]. Available: https://www.chromium.org/developers/design-documents/site-isolation

[18] P. Silva, R. Monge, and E. Fernandez B., "A pattern for Web Browser Infrastructure," *Pattern Language of Programs*, 2015.

[19] A. Barth, A. P. Felt, P. Saxena, and A. Boodman, "Protecting Browsers from Extension Vulnerabilities," *Ndss*, vol. 147, pp. 1315–1329, 2010. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.154.5579\&amp;rep=rep1\&amp;type=pdf

[20] N. Utakrit, "Review of Browser Extensions, a Man-in-the-Browser Phishing Techniques Targeting Bank Customers," *Proceedings of the 7th Australian Information Security Management Conference*, pp. 110–119, 2009. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-84864552184\&partnerID=40\&md5=3d08a9c7c4ba9dbe5e04fb831ad5257b$\backslash$nhttp://ro.ecu.edu.au/ism/19/

[21] T. Dougan and K. Curran, "Man in the Browser Attacks," *International Journal of Ambient Computing and Intelligence*, vol. 4, no. 1, pp. 29–39, 2012.

[22] "Hackers steal 150,000 us from mich. insurance firm." [Online]. Available: http://www.krebsonsecurity.com/2010/02/hackers-steal-150000-from-mich-insurance-firm/

[23] "Feds to charge alleged spyeye trojan author." [Online]. Available: http://krebsonsecurity.com/tag/spyeye/

[24] "Update [win] google chrome: Cross-site scripting - remote with user interaction." [Online]. Available: https://www.auscert.org.au/render.html?it=11648

[25] R. Colvin, "SmartScreen," 2010. [Online]. Available: http://blogs.msdn.com/b/ie/archive/2010/10/13/stranger-danger-introducing-smartscreen-application-reputation.aspx

[26] "Sandbox - The Chromium Projects." [Online]. Available: http://www.chromium.org/developers/design-documents/sandbox

[27] E. B. Fernandez, *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons, 2013.

[28] E. B. Fernandez, R. Monge, and K. Hashizume, "Building a security reference architecture for cloud systems," *Requirements Engineering*, Jan 2015. [Online]. Available: http://link.springer.com/10.1007/s00766-014-0218-7