A Misuse Pattern for Web Browsers: Interception of traffic

Paulina Siva[1], Raúl Monge[1] and Eduardo B. Fernandez[2]

[1]Departamento de Informática, Universidad Técnica Federico

[2]Department of Computer and Electrical Engineering and Computer Science, Florida Atlantic University

pasilva@alumnos.inf.utfsm.cl, rmonge@alumnos.inf.utfsm.cl, 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 an 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.*

Categories and Subject Descriptors

•Software Architectures → Patterns

General Terms

Design

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.

Social engineering is used as an umbrella term for a broad spectrum of computer exploits that employ a variety of attack vectors and attempts to psychologically manipulate a user. *Social engineering attacks* in [3] 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. This work attempts to define a Misuse Pattern that could appear when a social engineering attack succeeds; of course, is not the only way to make the misuse, but attackers can easily make it because the user of the browser is the *weakest link*.

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. The audience to which our paper is focused are browser developers, web application developers, researchers and for teaching purposes. Being the first two the most important, since they are the responsible for the implementation and correct use of secure mechanisms while a user is browsing in the Internet. Secure communication is an important matter, but in this work the focus will be on the browser and how to control the responses it receives.

2. Terminology

Current popular web browsers use a modular architecture in contrast with monoprocess architecture used in the nineties. Nevertheless, the core components of the browser are placed in a different manner, meanwhile others are replaced with new technology or deleted. In this paper we defined names for core components, but probably some of them may differ depending on implementation. We named Browser Kernel the component that represents the main process of a web browser responsible for receiving requests from the user using the web browser. A Controlled Process, in contrast, is a slave process of the Browser Kernel and is used to render the content for each request the user of the browser (a Browser User) needs.

3. Misuse Pattern: Interception of Traffic in the Web Browser

3.1 Intent

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

3.2 Context

A web browser fetches resources (web pages, services, etc.) from a Provider to satisfy the request 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 CSS. 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 Kernel that generated the request.

3.3 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 browser's processes, resulting in the interception of content that may change the Renderer or the browser itself. Depending on the type of attacker, it is possible that it may even affect the host where the browser is located.

The solution is influences by the following objectives:

- *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 compromise 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/forces:

- The Same Origin Policy (SOP) [7] is defined by the origin, and is used to separate different resources by its domain, scheme and port. It is the minimum security mechanism a browser has while requesting cross-origin resources, and divides the different kind of contents so they cannot interfere with each other. The SOP which every browser complies with, differs slightly from a web browser manufacturer to the other [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 Kernel, 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 right access control).
- Many browser manufacturers do not have robust defense mechanisms that allow an effective identification of malicious resources.

3.4 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 or scripts 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 some 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 Kernel, 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 Kernel, 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 Kernel. 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 dynamic language (based on ECMAscript) has a lot of reflection capabilities that can be used by a malicious program to achieve its purpose. A prototype language is a style of object-oriented programming in which behavior 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 monolithic architecture (monoprocess). Meaning that the browser is using the user’s permission to access directly the host resources.

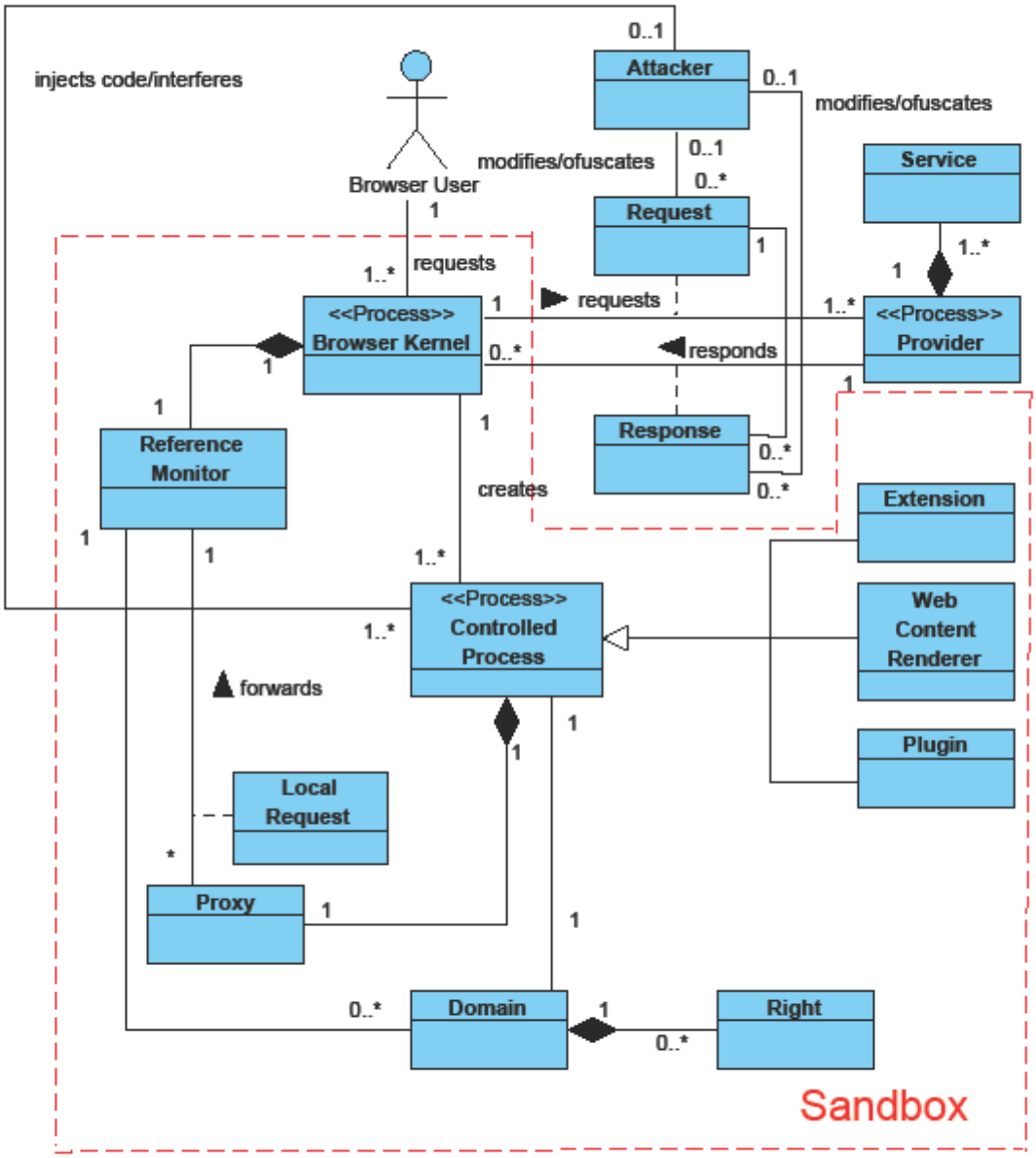

Figure 1. Class Diagram for the Misuse Pattern.

3.4.1 Structure

In Figure 1 the **Browser Kernel** is an entity that represents the main process of a web browser, which is constantly communicating with the host of the browser. A user who makes a **Request** to an Internet resources using a web browser, will be called **Browser User**. At the same time, a **Provider** is responsible for receiving external requests. According to the request, a **Provider** will send the **Service** (or resource) the **Browser User** needs in a **Response** message. Most Browsers use a central component to do operations that need to affect the Host of the Browser, a **Browser Kernel**. Figure 1 shows the Class diagram for the misuse pattern. For each new resource a **Browser Kernel** requests a **Web Content Renderer** instance (created or reused); this will inherit the **Controlled Process** properties and its methods. A **Plugin** and an **Extension** are elements that extend the functionality of the browser; the extender being for the exclusive use of the browser while the plugin can be used in other systems, such as the Adobe Reader Plugin. A **Sandbox** is a Controlled Execution Domain [19] created for a single **Controlled Process** instance. The **Sandbox** allows the process memory isolation and the access control of each communication between processes, such as an instance of a **Controlled Process** with the **Browser Kernel**; this applies to a **Web Content Renderer**, a **Plugin** and an **Extension** as well.

To communicate with the **Browser Kernel**, a **Proxy** created within a **Controlled Process** forwards a **Local Request** to the **Reference Monitor** inside the **Browser Kernel**. For every message sent from a **Controlled Process**, the **Reference Monitor** will check the **Domain**'s **Rights** to permit the access. The access control which the **Sandbox** delivers to each **Controlled Process** allows the isolation between different **Domains**. Depending on the manufacturer, a **Plugin** could not be sandboxed. 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 of it and the **Provider** (Figure 1). The attacker is able to intercept the messages between the **Browser Kernel** and **Controlled Process** using a malicious **Controlled Process**, which could be instanced as a **Plugin**, **Extension** or a **Web Content Renderer**.

3.4.2 Dynamics

In Figure 3 a series of required steps is shown, for one of the many misuses that can be made for the use case Make Request (Figure 2). The attacker is located between the Browser Kernel and the Controlled Processes, intercepting the original request or response and modifying the traffic to its taste; usually an attack based on this misuse is called Man-in-the-Browser (MITB) [15, 20, 21, 22]. This could also happen when the Browser User has allowed the installation of plugins, extensions or external programs in the Host and Browser Kernel.

Summary

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

Actor

Attacker

Preconditions

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

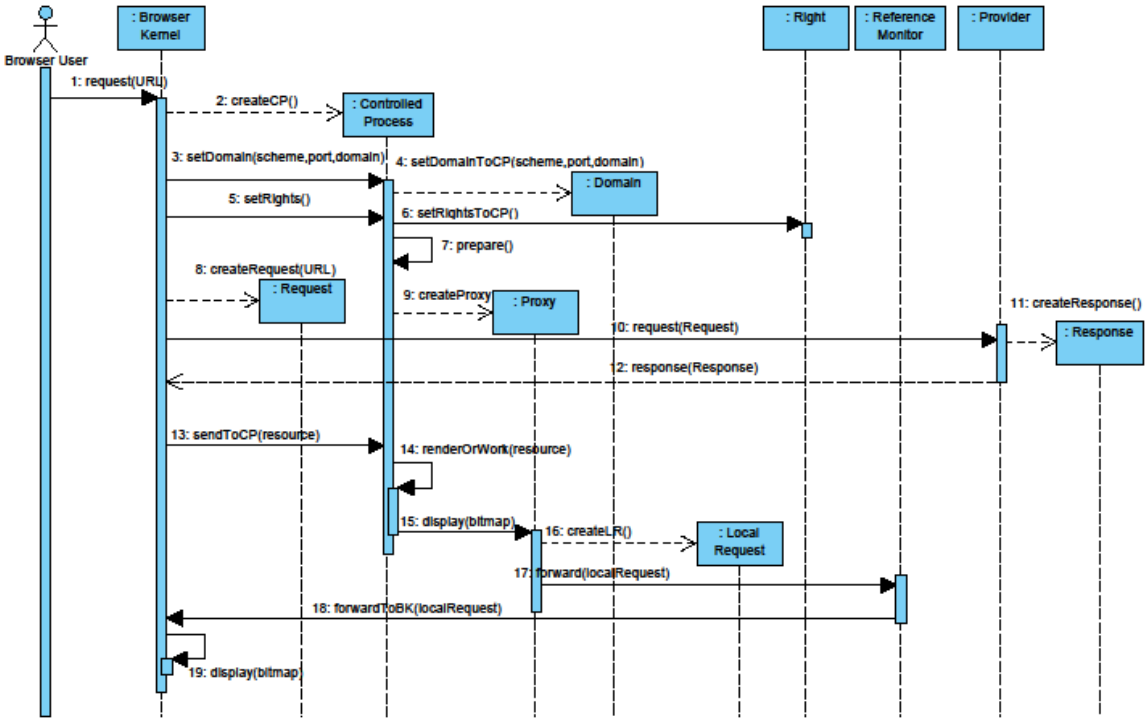

Figure 2. Sequence Diagram “Make Request" [18], with an already instanced Controlled Process. The first 9 steps are used by the misuse and then it starts the Partial Sequence Diagram for the misuse (Figure 3).

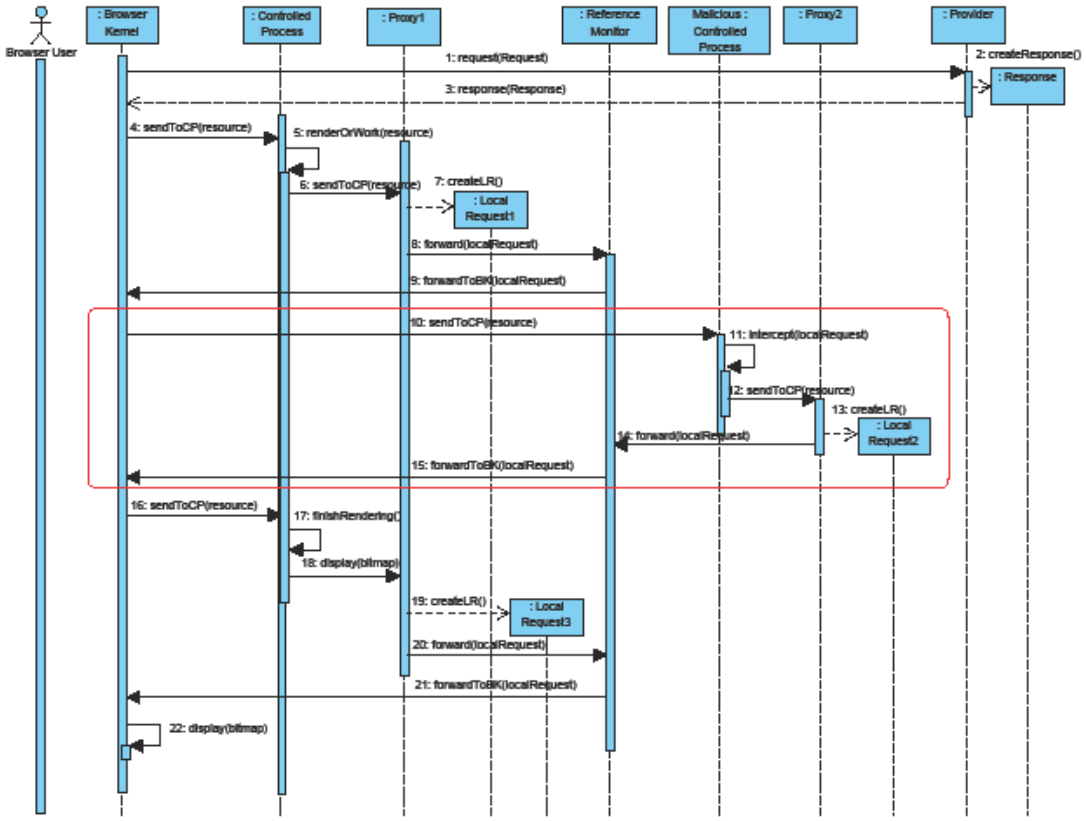

Figure 3. Partial Sequence Diagram for the misuse: Interception of traffic in the web browser (The first 9 step remain the same as the Make Request from [18], Figure 2).

Description

1. An attacker uses a social engineering technique or vulnerability in the system, and creates an entity between the Browser Kernel and Controlled Process and their communication channel, normally a plugin, binary or extension (but a malicious one). Also, an attacker could be using a *benign-but-buggy* extension or script to the same purpose. The social engineering attack starts as a normal “Make Request” use case (Figure 2), while the Browser User surfs the web and gets hocked by the attacker or clicks an URL inside a phishing e-mail.
2. A Browser User requires a browser to access a URL for some resource in a Provider, this is done by using a Browser Kernel already instanced in the Host. A Controlled Process and its Proxy are created or reused for rendering the resource needed. If the Controlled Process is new, the Domain and Rights of the Controlled Process are set. The first 9 step of the Make Request Sequence Diagram from [18] (Figure 2) are the same.
3. A Request is made from the Browser Kernel which creates a communication channel with the Provider, returning a Response after processing the request connection (steps 1 to 3 in Figure 3).
4. The Browser Kernel receives the Response, and forwards it to the Controlled Process which the original Request came from (step 4 in Figure 3).
5. From here the misuse starts, the original or “may be" tapped Response could follow different ways. While rendering the resource, somewhere in the Response obtained the need of calling a plugin, an extension or a program for the rendering process will appear, and an action from the attacker to intercept the messages will occur. Also, a Browser Kernel could have been tapped to perform that action and forward all the traffic to other Controlled Process; a malicious or to a *benign-but-buggy*. While sending the traffic, the Controlled Process needs to rely on its Proxy to send the data to the malicious plugin, extension or binary created through the Controlled Process. The traffic sent from the Controlled Process to the malicious one, has to make it through the Reference Monitor first, if the attacker wishes to be successful. The latter is done by sending a Local Request through the Proxy of the Controlled Process of origin, to the Reference Monitor in the Browser Kernel (steps 5 to 9 in Figure 3).
6. Then, the Browser Kernel sends the traffic to the malicious or to a *benign-but-buggy* Controlled Process and intercepts the traffic that is going to be used for rendering. Depending on the attacker's goals, the data can be spied or modified. Since the rendering process needs to finish, the intercepted traffic is returned through the Proxy in the malicious Controlled Process, to the Reference Monitor in the Browser Kernel, and then to the Controlled Process of origin to finish the rendering (steps 10 to 16 in Figure 3).
7. It can happen the traffic goes between the malicious Controlled Process and the Browser Kernel several times before it can reach to the Controlled Process, in charge of the rendering process.
8. Once the rendering process has finished, the bitmap is sent to the Browser Kernel (through the Proxy) and displayed in the Host. Also, the representation of the resource in the Web Content Renderer (DOM and data) could deviate from the original, thanks to the interception done by the attacker (steps 17 to 22 in Figure 3).

Post conditions

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.

3.5 Known Uses

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

- Zeus [23] 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.
- SpyEye [24] 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 Web Extension API could intercept the data before it reaches the Browser Kernel 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, 20]. 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.

3.6 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.

- If the browser manufacturers implement correctly the secure mechanism, the HTTP Only and Secure Cookie flags on the HTTP header could avoid the theft of session tokens, the use of cookies in scripts or the interception of data by a bystander.

3.7 Countermeasures

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

- Reputation services such as SmartScreen [26] 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 an hourly-basis.
- Browsers like Google Chrome and Internet Explorer offer Sandboxing [27, 14]. This defense mechanism limits the actions of the attacker which may affect the integrity of the system.

3.8 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.

3.9 Related Patterns

- The Web Browser Communication pattern [18].
- The Reified Reference Monitor [19], which describes how to enforce authorization rights when a subject requests access to a protected object or service and returns a decision (response).
- The Sandbox is another name for the pattern Controlled Execution Domain [19].
- Blacklist and Whitelist patterns [19]. The TLS pattern in [19] complements our pattern, for adding security to communications between Client and Server.

4. 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. The Web Browser Communication pattern is the main core of the Reference Architecture (RA) we are building. It represents the main component for a modular architecture of popular browsers like: Google Chrome/Chromium, Internet Explorer and Firefox. Other patterns related to Web Browser Communication pattern will be obtained in order to complete the Reference Architecture, such as the Web Content Renderer and Browser Kernel pattern. A work in parallel is the abstraction of defense mechanism as security patterns to build a Security Reference Architecture for web browsers.

We plan to build more Misuse Patterns for the Web Browser Communication 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].

Acknowledgements

We thank our shepherd, Yu Chin Cheng, for his useful comments that significantly improved the quality of the paper. We also thank Woei-Kae Chen for supervising our paper shepherding. This work was partially supported by CONICYT (grant Fondecyt 1140408).

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=10.1.1.295.6192n&rep=rep1n&type=pdf

[2] NSS Labs, “Evolutions In Browser Security," 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] R. Abrams, J. Pathak, and O. Barrera, “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-Subvertingn 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 Reference Architecture for web browsers: Part I, A pattern for Web Browser Communication," Proceedings of the 5th Asian Conference on Pattern Languages of Programs (AsianPLoP) 2016, Taipei, Taiwan, February 2016.

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

[20] 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=10.1.1.154.5579n&rep=rep1n&type=pdf

[21] 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-s2.0-84864552184n&partnerID=40n&md5=3d08a9c7c4ba9dbe5e04fb831ad5257b$nbackslash$nhttp://ro.ecu.edu.au/ism/19/

[22] 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.

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

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

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

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

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

[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-