

Browser Infrastructure Pattern

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 Fernandez
Department of Computer &
Electrical Engineering and
Computer Science
Florida Atlantic University
Florida, USA
ed@cse.fau.edu

ABSTRACT

Currently a lot of software developments create systems that are connected to the Internet, which allows to add functionality within a system and facilities to their *Stakeholders*. This leads to depend in a *web client*, as the *Web Browser*, which allows access to services, data or operations that the system delivers. Nevertheless, the Internet influences the attack surface of the new system, and unfortunately many stakeholders and developers are not aware of the risks they are exposed. The lack of Security Education in Software developers of a project, the low and scattered documentation of each browser (and standardization), could become a great flaw in big architectural developments which depends on the browser to do their services. A Reference Architecture of the *Web Browser*, using Architectural Patterns, could be a base for understanding the security mechanisms and its architecture, which interacts with a bigger web system. This would give an unification of ideas and terminology, giving a holistic view regardless the implementation details for both the browser and the system it communicates to. We developed a Browser Infrastructure Pattern which describes the infrastructure to allow the communication between a Web Client and Server in the Internet. With this work we propose an Architectural Pattern as the first piece of our Reference Architecture for the Web Browser.

Keywords

Web Browser, Web Client, Modular Architecture, Browser Architecture, Reference Architecture, Browser Infrastructure pattern

Introduction

Background

We present in this section patterns as well as their benefits. We also describe how to build more secure reference architectures by using security patterns

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

WOODSTOCK '97 El Paso, Texas USA

© 2015 ACM. ISBN 123-4567-24-567/08/06.

DOI: 10.475/123_4

Patterns are encapsulated solutions to recurrent problems and define a way to express requirements and solutions concisely, as well as providing a communication vocabulary for designers [11]. The description of architectures using patterns makes them easier to understand, provides guidelines for design and analysis, and can define a way of making their structure more secure.

Security patterns describe solutions to the problems of controlling (stopping or mitigating) a set of specific threats through some security mechanism, defined in a given context. The most common use of security patterns is to help application developers -who are not security experts- to add security in their designs. Patterns of this kind also are used to reinforce a legacy system.

The aim of a Reference Architecture is to provide a guide for developer, who are non security experts, in the development of Architectures for concrete versions of the system or to extend it. With the use of Architectural Patterns we describe the Browser Architecture as a Reference Architecture (RA). A RA is created by capturing the essentials of existing architectures and by taking into account future needs and opportunities, ranging from specific technologies, patterns and business models. It can also be derived from domain models.

A Secure Reference Architecture is a Reference Architecture where security services have been added in appropriate places to provide some degree of security for a system environment. The basic approach that we will use to build a Secure Reference Architecture is by applying a systematic methodology from [7, 9], which can be used as a guideline to build secure web browsers systems and/or to evaluate their security levels. We started to build a Reference Architecture as a first step, in a student work, and now we are trying to improve it using security patterns and misuse patterns. By checking if a threat, expressed as a misuse pattern, can be stopped or mitigated in the secure reference architecture, we can evaluate its level of security.

In this work, a Browser Infrastructure Pattern is presented as a first step in to the process of developing a Secure Reference Architecture for the Web Browser. Threat analysis and security patterns was done in our previous student work (graduation report), and we will improve it in the construction of the SRA.

Related Work

Reference Architecture of the Browser

We tried to find studies by searching relevant keywords in Scopus or doing a forward snowballing from a work [14] we

knew. Unfortunately to the date there are a few works related to the construction of a Reference Architecture for the Web Browser.

In the study made by Larrondo-Petrie et. at [15] a web browser analysis is done with the goal of obtaining a Domain Model, and Object Model and a Feature Tree which described the structure and functionality a browser had. The domain, explained in th paper, is a distintive set of objects which act according to rules and policies that characterize the Domain. The used methodology obtains the domain is called Object Oriented Analysis. To identify

FALTA TRADUCIR ESTO!!!

El dominio, según explica el trabajo, es un set distintivo de objetos que se comportan de acuerdo a reglas y política que caracterizan el Dominio. El Análisis de Dominio es realizado para identificar dominios y cómo éstos interactúan con otros. La metodología usada para obtener los Dominios es el *Object Oriented Analysis*. Además de identificar, se clasifican estos dominios de acuerdo a su rol en el sistema terminado como: Dominio de Aplicación, Dominio de Servicio, Dominio de Arquitectura y Dominio de Implementación. El Modelo de Objetos sirve para entregar más detalles, un resumen general de las Entidades del *Web Browser* y sus relaciones. El *Feature Tree* pretende entregar detalles sobre los aspectos funcionales de la aplicación. El Modelo planteado, según el artículo, debería ser útil para los Desarrolladores de Software que construyen **Aplicaciones Web basadas en el uso del Browser**. Este estudio se encuentra bastante lejos de lo que se quiere hacer en este trabajo, pero sirve para obtener un transfondo de lo que sucede en el *Web Browser*, aún cuando la información esté muy desactualizada.

En el trabajo de Grosskurth et al. [14, 13] se utiliza una herramienta de ingeniería inversa, para obtener una arquitectura de referencia de muy alto nivel en base a dos navegadores open-source: Mozilla y Konqueror. Lo obtenido captura los subsistemas fundamentales comunes a los sistemas del mismo dominio, así como las relaciones entre estos subsistemas. En esta arquitectura se identifican los siguientes subcomponentes: Interfaz Usaria, Persistencia de Datos, Browser Engine, Rendering Engine, Networking, Interprete de Javascript, XML Parser y Display Backend. Se menciona que estos componentes están estrechamente integrados (high coupling) con el Rendering Engine, lo cual tiene sentido en la arquitectura monoproceso que poseen Mozilla y Konqueror; es una decisión de diseño muy común en los *Browser* de la época. Al identificar estos componentes, se comenta que esto podría servir tanto en el diseño y durante la mantención de un sistema, pues mejora el entendimiento de ésta al ayudar a analizar los trade-off entre diferentes opciones de diseño; o también puede servir como un *template* para obtener nuevos diseños. Una vez obtenida la arquitectura conceptual, se inició una evaluación de ésta al comparar las arquitecturas concretas de cada browser open-source, extraídas desde el código fuente, para ver qué tanto el modelo conceptual era cercano a la realidad; la constante comparación permitió además refinar la Arquitectura de Referencia. Los browsers usados para validar fueron: Epiphany, Safari, Lynx, Mosaic y Firefox. Si bien la arquitectura presentada entrega bastante información a alto nivel, no desarrolla más que esa capa de abstracción, además parece ser que depende también de la implementación usada en la herramienta de ingeniería inversa.

En el documento [12] realizado en el año 2000, se describe

la experiencia realizada al extender el trabajo del proyecto TAXFORM. Usando PBS, una herramienta de Ingeniería Inversa, se extrajo la arquitectura de software del navegador Mozilla, con el objetivo de entender la estructuración de sus componentes; además de crear vistas arquitecturales de alto nivel del sistema. El modelo arquitectónico obtenido contiene 11 subsistemas de alto nivel, de éstos los que más se destacaron fue el *HTML Layout*, la implementación de herramientas y el código de la interfaz de usuario. En el año en que se lleva a cabo este estudio (2000), se menciona que la arquitectura ha decaído significativamente en muy poco tiempo, o su arquitectura no fue planificada cuidadosamente desde el comienzo; parte de lo anterior, el autor cree que es secuela de la *Guerra de Navegadores*. Si bien el trabajo ayuda a entender un poco la estructura detrás del navegador, este trabajo es muy antiguo y la versión más actual del navegador ha cambiado bastante. Además, lamentablemente el enfoque de este estudio no es intentar entender lo que hace cada subsistema, si no que es la implementación de la herramienta misma para obtener la arquitectura de software del browser seleccionado.

En [16] se propone un *Browser* llamado Anfel SOFT, donde gracias al uso de Inteligencia Artificial, crea agentes que permiten mejorar la experiencia del usuario. El trabajo asegura que el browser será capaz de aprender el comportamiento de navegación del usuario, y guiará al usuario en su navegación para que ésta sea lo más efectiva posible. El paper obtiene los subsistemas que se pueden encontrar en un browser de la misma manera que lo realiza [14]. Si bien la arquitectura que muestra refleja parte de lo visto en los 3 browsers escogidos en este estudio, no da detalles acerca de cada subsistema identificado. Además la Arquitectura de Referencia que entrega es la misma vista en [14, 13], y a pesar que identifica otros posibles componentes, no agrega nada nuevo.

Podemos ver en los trabajos que algunos construyen una Arquitectura de Referencia basada en técnicas de Ingeniería Inversa. En cada uno de ellos el trabajo ha sido a muy alto nivel y la descripción de los subcomponentes del sistema es mínima. Si bien explican las relaciones entre éstos, no dan un mayor entendimiento en cómo se comportan en ciertas situaciones. En este trabajo se espera profundizar un poco más en la abstracción obtenida, incluyendo información de tanto los casos de uso del *Browser* como las actividades que se realizan con otros usuarios. Desafortunadamente para esta memoria, no existe mucha literatura sobre el desarrollo de una Arquitectura de Referencia del *Browser*, y de lo que hay, el trabajo más actual es el realizado por [16] en el año 2009.

Secure Software Development

La literatura que habla de la construcción de *Secure Software* o Software Seguro, indica que los practicantes de Desarrollo de Software deben entender, en gran medida, los problemas de seguridad que podrían llegar a ocurrir en sus sistemas. No basta con saber cómo unir las piezas, no basta con que cada pieza de por sí sea segura, si los componentes del sistema no actúan de forma coordinada, probablemente éste no será seguro [10], dado que la seguridad es una Propiedad Sistémica que necesita ser vista de manera holística y al inicio del proceso.

Browser Infrastructure Pattern

Intent

The Browser Infrastructure Pattern allows the request of a web resource in the Internet to a **Browser User** (BU), which is a user who uses a Browser within a **Host**. The Pattern lets visualize the communication between the components that make the Web Browser and the Provider (i.e, a Server), to whom the request is made.

Example

Within the Host it is possible a lack of resources that a Host user may need. The request of external services or resources is the main reason of the Internet existence. This kind of task it is possible to do in a lot of ways, it all depends on what the Provider wish to deliver to others.

Context

Browser User is a Host user which uses a Browser, and the Provider is an entity which can be accessed in the Internet. The contact between each other is normally done by Web Applications or Servers which communicates using the HTTP protocol. A Browser let the Browser User access and visualize the external resources a Provider has and a Browser User may need.

Problem

Some Browser Users could need resources from a Provider, but the user maybe will need them in a special format or they should be presented in the screen of the computer to be visualized. In this case, if an appropriate tool is not used, the resource could not be helpful if it can not be used correctly. How can the Host and Provider be prepared to this situation? The solution to this problem must resolve the following problems:

- Transparency: the user behind the Host should not be worried of what it is done, while a request to a Provider has been issued.
- Stability: The *Browser* must be capable of working, even if a web page had a problem to be seen or there is an internal problem.
- Isolation: Each *request* must not interrupt others.
- Heterogeneity: It does not matter the type of Provider to which the Browser communicates, it should be possible to interact with whatever type it is, and also it should be capable to show adequately the content of the obtained resource.
- Availability: The user of the Host, should be capable to request at any time.

Solution

A *Web Browser* can satisfy the request a user of the Host by the Browser User, either by one or more instances of the Browser User, which allows for a variety of options to browse in the internet. A Browser must be able to deliver a fast and stable navigation, without affecting each accessed sites.

Structure

The Browser Client (BC) is an entity that represents the main process of a Web Browser and comprises the minimum number of components which constitute a Browser. A Host (H) houses and interacts with the BC. H is composed mainly by Hardware (HW) and a Operative System (OS). At the same time, a Provider (P) has also HW and OS, but additionally has a Web Server (WS) which is responsible for receiving external requests. Browser Client (BC), Sandbox (S), GPU Instance (GPUI) y Plugin are instances of Process (Pr), which resides within a H. Most Browsers use a central component to do operations that need to affect the Host of the Browser. Figure 1 shows the Class diagram for the Browser Infrastructure Pattern. For each resource a BC requests, a Sandbox hosting each Web Content Renderer (WBR) instance created will allow the browsing and visualization of the resource obtained. The BC component acts as a broker for the requests coming from the Sandbox (which host a WBR), this allows a fine control over the sent messages (using PC/IPDL/COM) between communicating process. If there is a need for the H resources, a GPU Instance and Plugins are elements which need to communicate directly with the Sandbox, which in its instead will ask to the BC for resources. A user who makes a request to a Internet resources using a Web Browser, will be called Browser User (BU). BU uses BC to make requests to one or more Providers, where the latter uses a Web Server (WS) to receive requests and reply back (Figura 1).

Dynamics

Some use cases are the following:

- Make Request (actor: Browser User)
- Cancel Request (actor: Browser User)
- Save Resource (actor: Browser User)
- Receive Request (actor: Provider)
- Ask for Resources (actor: Host)

We show in detail Make Request below. (Figura 2):

Summary

A Browser User needs a URL resource which can be obtained by using the HTTP protocol, as required by the Provider. The Browser Client will be used by a User Browser to perform the display of the URL resource.

Actor

Browser User

Preconditions

The Host must have one or more Browser Client for the Host user. In addition to being connected to a network or the Internet. The Provider you want to contact must also be available.

Description

Note: Messages between the Browser Client and Sandbox can be both synchronous and asynchronous [4, 2] . We not specify in great detail, because what matters in this work will be the origin and destination of the message (is not within the scope to see synchronization).

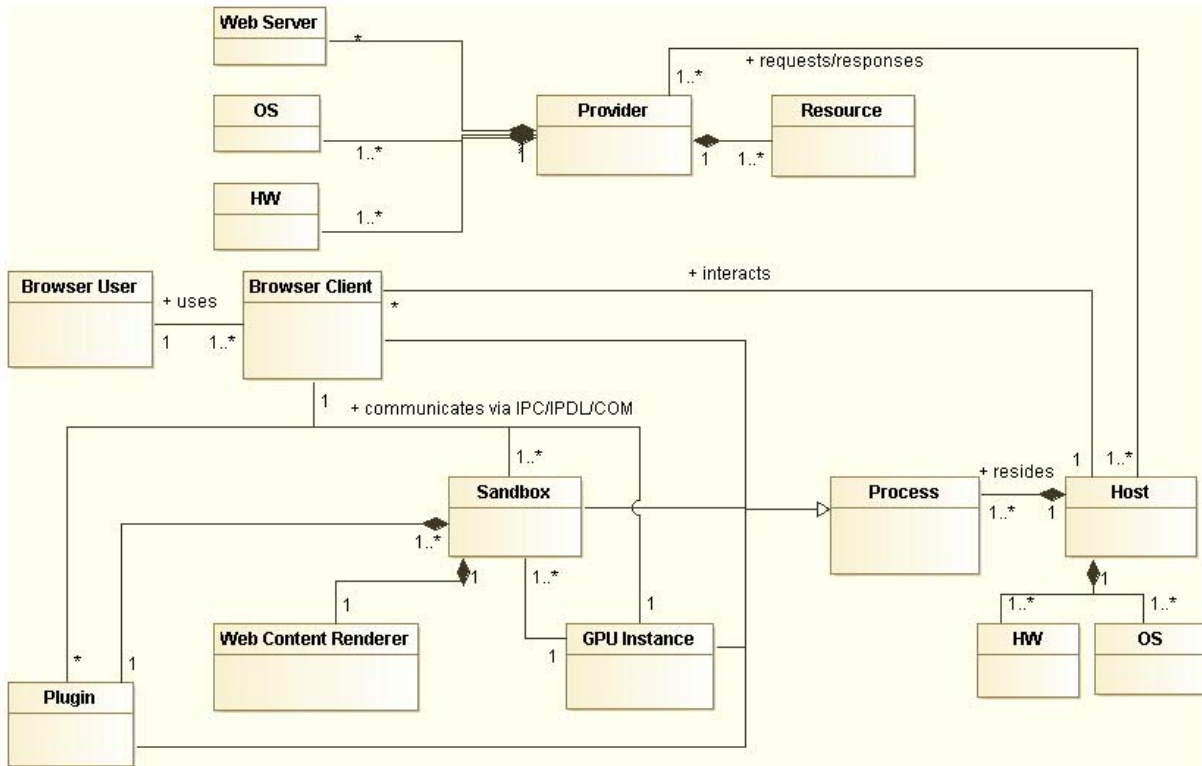


Figure 1: High-level Componentets of the *Browser*.

1. A Browser User requires a browser to access a URL for some resource in a Provider, this is done by using an already instanced Browser Client in the Host. Inside the Sandbox there is an instance of Web Content Renderer pattern.
2. 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.
3. The Browser Client receives the request, and verifies through its policy engine if the Sandbox action is allowed.
4. If the Sandbox action is permitted, the Network API within the Browser Client, to obtain a host resources (via system calls), it is used. The Browser Client communicates internally with the Host, and the latter must review its policies to ensure that the Browser Client has the privilege of making a request to the Host resources.
5. If access to the resource is allowed, the Browser Client may *request* through the Network API. If the *request* is not a *pre-flight*, the Provider will receive the *request* and work on it.
6. The Provider will send a *response* to the *request* received. Depending on how it is implemented the Browser Client, it may or not have to wait for the response (synchronous or asynchronous) of the Provider.
7. Once the response obtained it is stored in the cache, unless directed to do other way so.
8. The response to the *request* is sent by a communication channel to the Sandbox which originated and then the Web Content Renderer. If a response was received by the *request*, the Web Content Renderer is ready to prepare the parsing of the website or use a plugin or GPU to support the display of the resources obtained by the URL. Otherwise, the Web Content Renderer within the Sandbox will create an error page.
9. The *Renderer* obtains a bitmap to be sent to the Client Browser, so that the Host can present it. Before doing this, the BC should check that the Sandbox which host the Web Content Renderer possess the permissions to do so.
10. If the permissions are sufficient, the Browser Client sends the bitmap, as a parameter, in the system call made to the Host. Finally, H must check that the system call made by the Browser Client has the required permissions.

Curso Alternativo

- El Provider no esté disponible.
- El recurso al que apunta la URL no exista.
- Se cancela el *request* realizado.

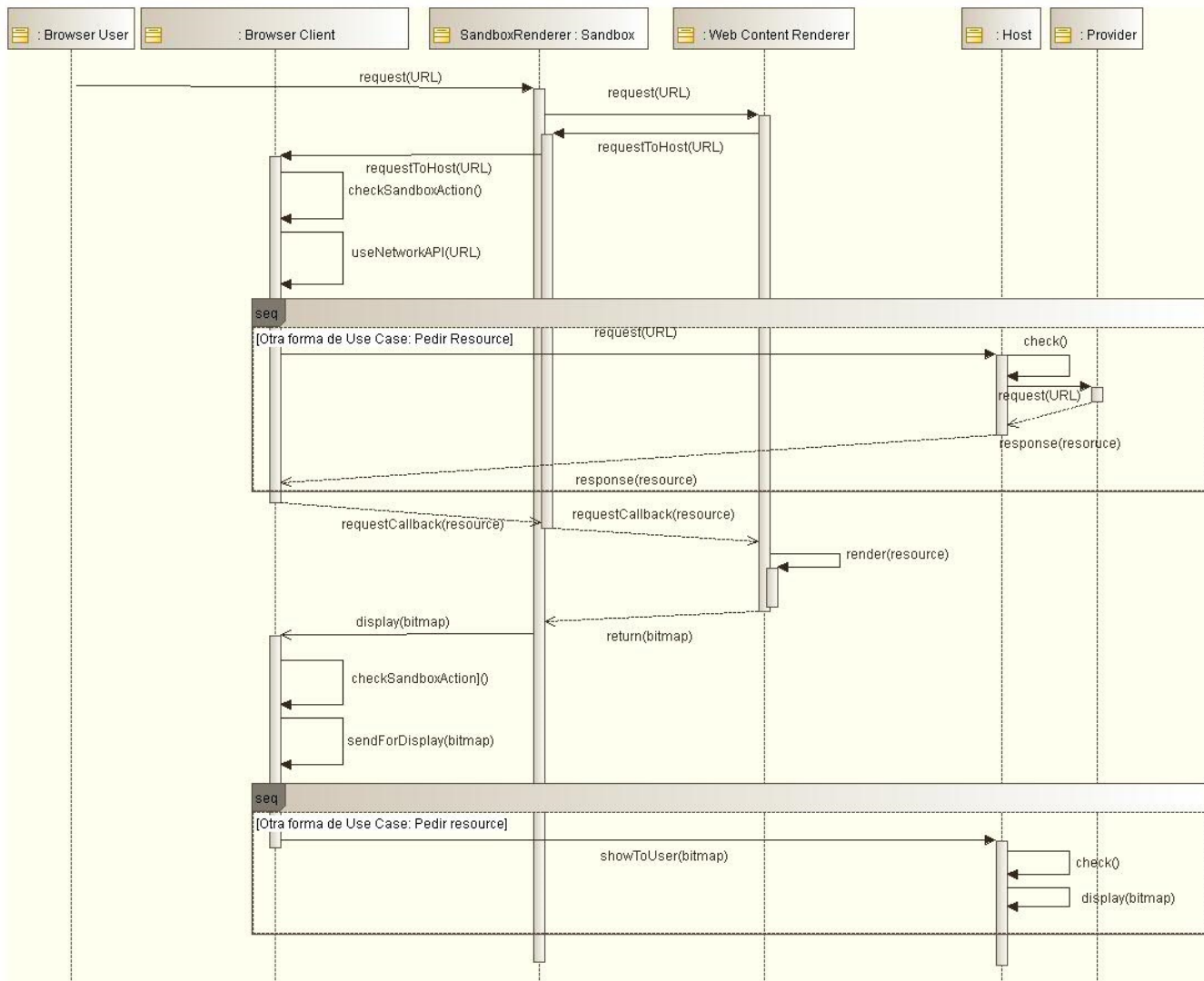


Figure 2: Diagrama de Secuencia: Realizar Request.

Condiciones póstumas

El *Browser* recibe el recurso indicado por la URL y se muestra por el periférico la salida del recurso al usuario del Host.

Implementation

- El Sandbox puede ser implementado de diversas maneras. Google Chrome [5] se basa en no reinventar la rueda y utiliza los mecanismos de protección que el Sistema Operativo (por ejemplo: Windows o Linux) del Host ya tiene incorporados para proteger al usuario, evitando que el proceso no tenga acceso al sistema de archivos y teniendo una API de llamadas al sistema muy restrictivas en el Web Content Renderer. Google Chrome, Firefox e Internet Explorer asumen que los Sandboxes son procesos que deben registrarse bajo el principio de menor privilegio (least privilege). La mínima configuración del Sandbox se compone de 2 procesos: El proceso privilegiado o Broker que es representado por el Browser Client, y el o los procesos que están bajo el Sandbox o targets.

- Para hacer cumplir con el Same Origin Policy, Google Chrome, Firefox e Internet Explorer utilizan diferentes esquemas; por ejemplo: Google Chrome deja el trabajo a su Renderer (Web Content Renderer en este caso) para dejar aisladas las páginas/recursos de diversos sitios.

Consequences

El patrón Browser Infrastructure provee los siguientes beneficios:

- Transparencia: La navegación del usuario se realizará casi de manera automática, solo en casos muy puntuales el usuario tendrá que tomar una decisión sobre el recurso que desea pedir.
- Estabilidad: Gracias a que el Browser Client, Sandbox, Plugin y GPU Instance son procesos del Host independientes, el fallo de uno no generará problemas en el otro (crash, corrupción de memoria, etc).

- Aislación: Dependiendo del tipo de aislación es posible separar las distintas peticiones, de manera que no interfieran entre sí, a menos que se desee.
- Heterogeneidad: Dado que cada Browser Client trata de seguir los estándares de la W3C, toda página que también siga estas guías podrá ser visualizada, así como también otro tipo de recursos.
- Disponibilidad: Cada proceso es independiente y posee sus propias hebras de ejecución, y fueron creadas específicamente para que la interfaz de usuario pueda mantenerse fluida.

Al mismo tiempo, el patrón posee las siguientes debilidades:

- Dado que se inician procesos independientes para navegar a un recurso (dependiendo del esquema que utilice el browser), es posible que una gran cantidad de recursos se vayan a usar para mantener todo abierto.
- Aquellos Provider que no hayan cumplido con las especificaciones de la W3C, mostrarán su resource de forma incorrecta por el *Web Browser*.

Example Resolved

Con el patrón entregado ahora es posible poder navegar de forma fluida a todos los recursos en la Internet que deseamos. Es posible proveer a través de la aislación de los componentes: rapidez, seguridad y estabilidad. El Browser User solamente se molestará en la navegación, cuando se requiera de su autorización para ingresar a ciertos recursos del Host que sean privilegiados, como el sistema de archivos. Cada usuario del Host, puede utilizar el Browser Client que desee, dado que cada uno de ellos es aislado por procesos independientes, así también los otros Browser Clients entre sí.

Known Uses

- Actualmente, la separación de los componentes del *Browser* en varios procesos, con diferentes niveles de acceso, se conoce como una Arquitectura Modular [17]. Esto permite la separación de preocupaciones del navegador, lo que se traduce en una mayor estabilidad, aislación, seguridad y rapidez.
- Google Chrome se basa en la arquitectura modular, donde cada proceso Renderer se comunica con el Browser Kernel [3]. Internet Explorer, por ser privativo no da mucha información sobre su estructura o detalles de su implementación; [6] sobre la arquitectura Loosely-Coupled [1] y sus componentes, pero sin entrar tanto en detalle. Firefox, por su parte posee las dos implementaciones: monoprocesos y multiproceso. Electrolisis es el nombre de la arquitectura modular que se está implementando, pero que aún no ha sido finalizada completamente.

Related Patterns

- El patrón Web Content Renderer, el cual está en proceso de desarrollo, representa el subsistema dentro del Sandbox que permite realizar el parsing del *resource* o recurso obtenido por medio de una petición.
- El patrón Browser Kernel, también en desarrollo, representa el subsistema que representa el navegador Web.

Este componente actúa como un Reference Monitor [10] para todas las solicitudes que el Renderer llegara a necesitar.

- El Sandbox se comunica con el Browser Client siguiendo un patrón como el que indica el Policy Authorization [10]. Por cada petición del Renderer, el Sandbox tendrá el acceso si las políticas que el Browser Client permiten tal acción.

Conclusions

Un navegador Web pareciera ser un Software de mediana complejidad para tanto usuarios como desarrolladores sin experiencia en seguridad, pero lamentablemente esta pieza de Software permite realizar una variedad de vectores de ataque, tanto en un usuario usándolo como en el sistema con el que interactúa. Por lo tanto es importante comprender su estructura y como éste interactúa con Stakeholders internos como externos.

Es de esperarse que en el futuro la mayoría de los *Web Browser* tomen forma de una Arquitectura Modular. Por lo tanto, es importante que los desarrolladores conozcan los procesos internos del *Web Browser* al momento de desarrollar un sistema que se comunicará con éste. Tanto la Arquitectura de Referencia como el patrón de Mal Uso presentados, apuntan a entregar el conocimiento básico de los componentes e interacciones entre el *Web Browser* y un proveedor de recursos externo; así como también de las amenazas que existen.

Construcción del primer Patrón Arquitectural sobre la infraestructura del *Web Browser*, para poder entender de manera holística los componentes, interacciones y relaciones. Una parte de la Arquitectura de Referencia ha sido construida, a través de la abstracción del Patrón Browser Infrastructure. Además se ha conseguido caracterizar los Stakeholders y uno de los casos de uso más importantes. De lo que tenemos por conocido, esta es la segunda Arquitectura de Referencia construida del *Browser*.

El trabajo propuesto permite comprender mejor, por medio de la Arquitectura de Referencia parcialmente construida, tanto componentes como amenazas existentes. Además como no está sujeto a implementaciones específicas, es posible generalizar ciertos resultados en otros Browsers.

Future Work

El trabajo futuro que se realizará para obtener el grado de Magister, irá relacionado a la creación de una Arquitectura de Referencia de seguridad del *Web Browser*, utilizando la misma metodología presentada acá. Otros patrones relacionados al Patrón Browser Infrastructure serán obtenidos para así completar la AR ya iniciada, como por ejemplo el patrón Web Content Renderer y Browser Kernel. Un ejemplo del tipo de trabajo que se pretende realizar puede ser visto en [8], donde este estudio realiza actividades para poder construir software seguro y evaluar los niveles de seguridad de un sistema ya construido.

Se planea construir más Patrones de Mal Uso, para el Patrón Browser Infrastructure para continuar con el estudio de amenazas posibles dentro del *Browser*, como una manera de educar a los Desarrolladores y Stakeholders de los peligros existentes. Al mismo tiempo estos patrones permitirán la construcción de esta AR de seguridad. En esta

misma dirección, además de encontrar las amenazas posibles de existir en el sistema, se necesita encontrar las contramedidas o defensas de seguridad que permitan evitar o prever esas amenazas a través de Patrones de Seguridad sobre la Arquitectura de Referencia construida. Lo anterior es posible de realizar bajo el mismo ejercicio ya realizado en este trabajo, buscando amenazas sobre cada acción realizada en cada Caso de Uso del navegador.

En cuanto a los *Web Browser*, los ataques basados en Ingeniería Social parece que no disminuirán en un buen tiempo, pues no existen tecnologías actuales que puedan detectar en un 100% y sin falsos positivos los posibles peligros que pueden traer. Tecnologías como CAMP (Content-Agnostic Malware Protection) parecen ser parte de la solución, pero aún están lejos de ser perfectos.

1. REFERENCES

- [1] IE8 and Loosely-Coupled IE (LCIE) - IEBlog - Site Home.
- [2] Inter-process Communication (IPC) - The Chromium Projects.
- [3] Multi-process Architecture - The Chromium Projects.
- [4] Multiprocess Firefox | Bill McCloskey's Blog on WordPress.com.
- [5] Sandbox - The Chromium Projects.
- [6] M. Crowley. *Pro Internet Explorer 8 & 9 Development: Developing Powerful Applications for The Next Generation of IE*. Apress, Berkely, CA, USA, 1st edition, 2010.
- [7] E. Fernandez, M. Larrondo-Petrie, T. Sorgente, and M. VanHilst. A methodology to develop secure systems using patterns. *Chapter*, 5:107–126, 2006.
- [8] E. B. Fernandez, R. Monge, and K. Hashizume. Building a security reference architecture for cloud systems. In *Proceedings of the WICSA 2014 Companion Volume*, page 3. ACM, 2014.
- [9] E. B. Fernandez, N. Yoshioka, H. Washizaki, J. Jurjens, M. VanHilst, and G. Pernu. *Using Security Patterns to Develop Secure Systems*. IGI Global, 2011.
- [10] E. Fernandez-Buglioni. *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons, 2013.
- [11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.
- [12] M. W. Godfrey and E. H. S. Lee. Secrets from the Monster: Extracting Mozilla's Software Architecture. In *Proc. of 2000 Intl. Symposium on Constructing software engineering tools (CoSET 2000)*, pages 15–23, 2000.
- [13] A. Grosskurth and M. W. Godfrey. Architecture and evolution of the modern web browser. URL: <http://grosskurth.ca/papers.html#browser-archevol>. Note: submitted for publication.
- [14] A. Grosskurth and M. W. Godfrey. A reference architecture for web browsers. pages 661–664, 2005. URL: <http://grosskurth.ca/papers.html#browser-refarch>.
- [15] M. Larrondo-Petrie, K. Nair, and G. Raghavan. A domain analysis of web browser architectures, languages and features. In *Southcon/96. Conference Record*, pages 168–174, Jun 1996.
- [16] A. Systems and N. Lwin. Agent Based Web Browser. *2009 Fifth International Conference on Autonomic and Autonomous Systems*, 2009.
- [17] T. Vrbanec. The evolution of web browser architecture. pages 472–480, 2013.