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 Infrastrcture 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

WOODSTOCK '97 El Paso, Texas USA

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 securityx

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 Browsr. Threat analysis and security patterns was done in our previous student work (graduation report), and we will improve it in the construtions 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 Analyis. 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 interactuan con otros. La metodología usada para obtener los Dominios es el *Object Oriented Analyis*. 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, ségún 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 Usuaria, 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 ademas 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 subcomponetes 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 si sea segura, si los componentes del sistema no actuan 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 communicats 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 Providr, but the user maybe will need them in a special format o they should be presented in the screen of the computer to be visualized. In this case, if an appropiate tools 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, evn 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 communicats, it should be possible to interact with whatever type is it, 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 requst 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 accesed sites.

Structure

The Browser Client (BC) is an entity that represents the main process of a Web Browser and comprises the minimum number of componenets which constitute a Browser. A Host (H) houses and interacts with the BC. H is composed mainly by Hardware (HW) and a Oprative System (OS). At the same time, a Provider (P) has also HW and SO, but additionally has a Web Server (WS) which is responsabile 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 Patter. For each resource a BC rquests, a Sandbox hosting each Web Content Renderer (WBR) instance created will allow the browsing and visualization of the resource obtained. The BC componenet acts as a broker for the requests comming 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 o 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 Resoruce (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).

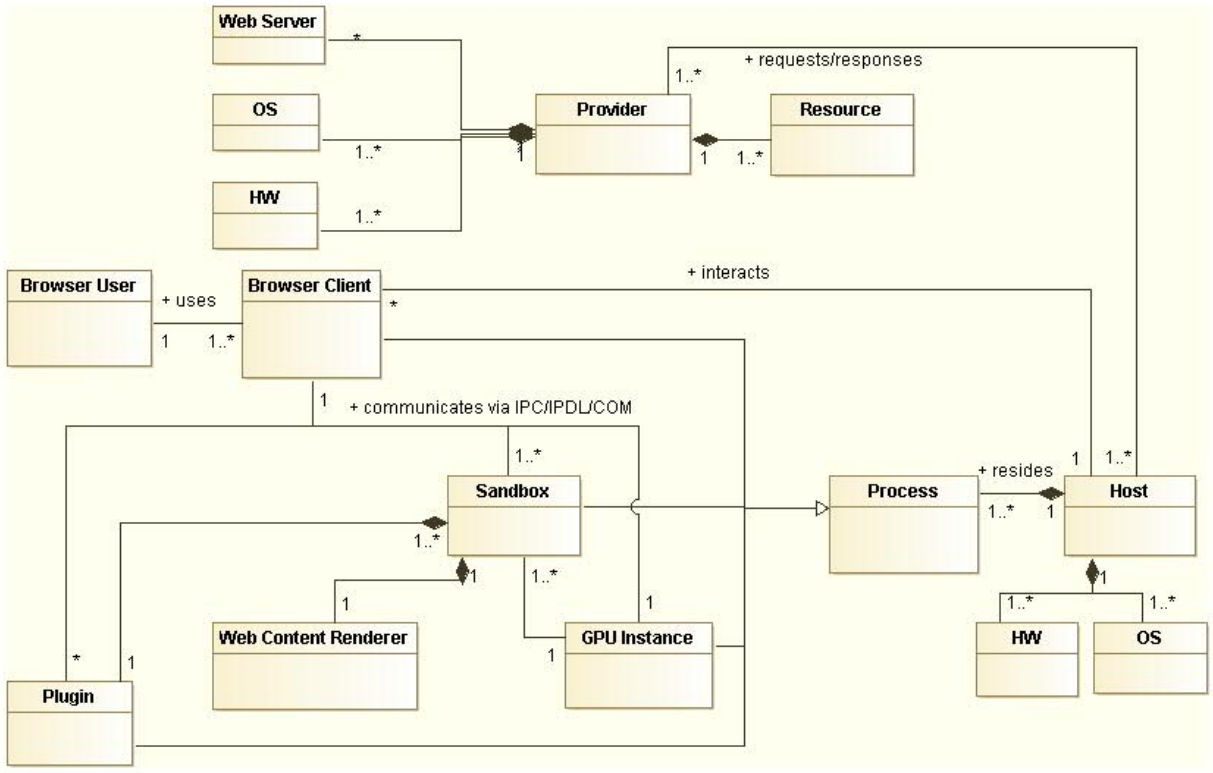

Figure 1: High-level Componenets of the *Browser*.

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.

Alternative flow

- The Provider is not available.
- The resource pointed by the URL does not exists.
- The request is cancelled.

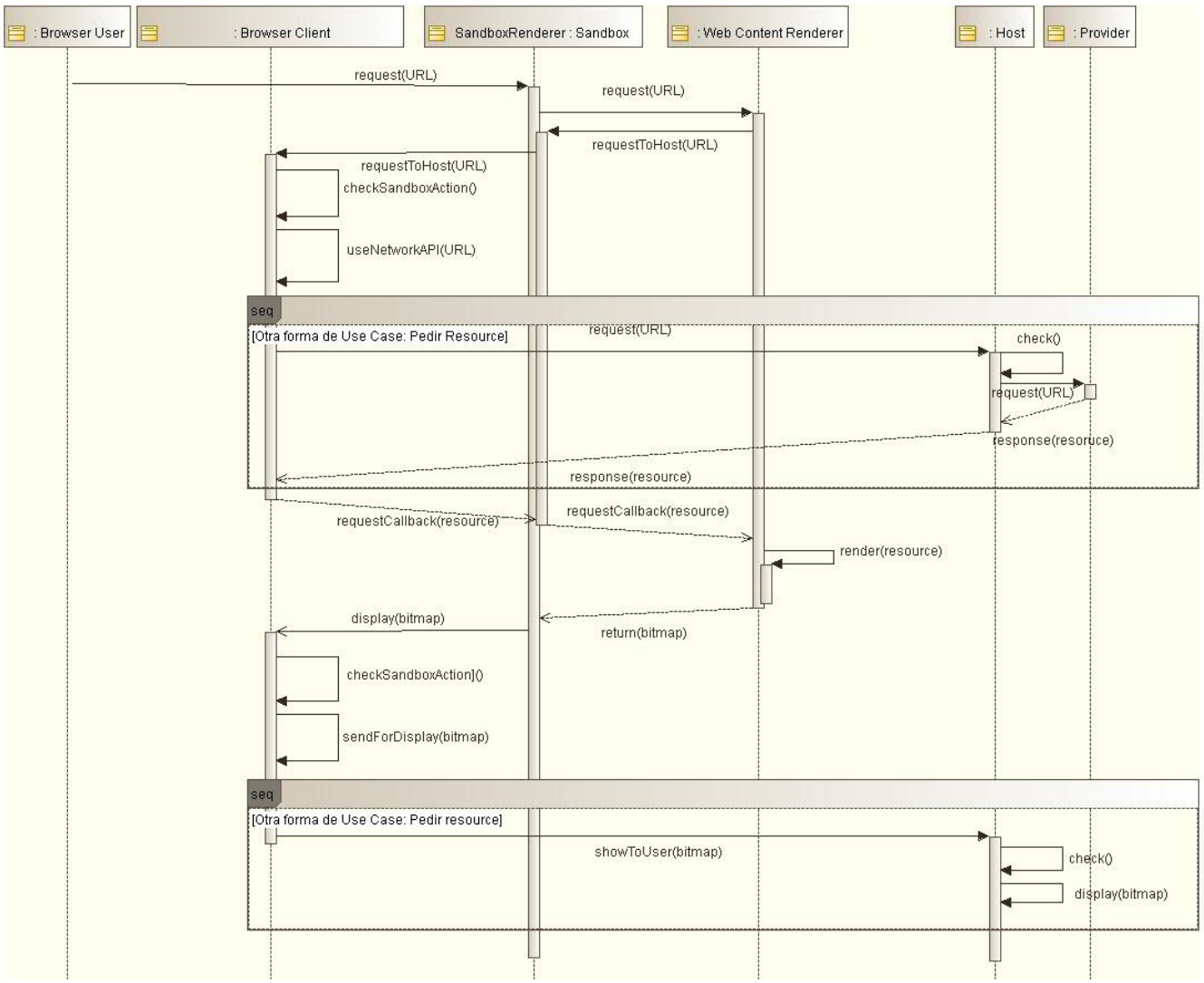

Figure 2: Diagrama de Secuencia: Realizar Request.

Postconditions

The *Browser* receives the resource indicated by the URL and is displayed by the peripheral device output to the Host user.

Implementation

- The sandbox may be implemented in various ways. Google Chrome [5] is based on not reinventing the wheel and use the protection mechanisms which offers the OS (e.g, Windows or Linux) Host to protect the user. This prevents to any process the access to the file system, and having an restrictive API in the web Content Renderer. Google Chrome, Firefox and Internet Explorer assume that Sandboxs are processes that should be governed by the principle of least privilege (least privilege). The minimum configuration Sandbox include 2 processes: The privileged process or Broker who is represented by the Browser Client, and the processes hosted by Sandboxes or targets.
- To enforce the Same Origin Policy, Google Chrome, Firefox and Internet Explorer use different schemes; for example: Google Chrome leaves his work to the Renderer (Web Content Renderer in this case) to leave isolated pages/resources from various sites.

Consequences

The Browser Infrastructure pattern provides the following benefits:

- Transparency: The user navigation is done almost automatically, only in rare cases the user will have to make a decision on the resource asked.
- Stability: Because the Browser Client, Sandbox, and GPU Instance Plugin are independent Host processes, the failure of one will not generate problems in other (crash, memory corruption, etc.).
- Isolation: Depending on the type of isolation you can separate the different requests, so they do not interfere with each other, unless it is desired.

- Heterogeneity: Because each Browser Client tries to follow the standards of the W3C [**?**], every page that follows these guidelines can be viewed, as well as other resources.

- Availability: Each process is independent and has its own thread of execution, these were specifically created to help to maintain the User Interface smooth.

At the sam time, this pattern has the following liabilities:

- Since independent processes are used to browse a resource (depending on the scheme using the browser), it is possible that a lot of resources of the Host are to be used to keep everything open.

- Provider who have not met the specifications of the W3C, their resource ar displayed incorrectly by the Web Browser.

Example Resolved

With the given pattern it is now possible to navigate smoothly to all resources on the Internet we want. It is possible to provide through the isolation of the components: speed, security and stability. The Browser User will only concern about the navigation, unless it is required for its explicit permission to enter certain Host resources that are privileged (e.g, the file system). Each Host user can use the Browser Client they want, because each one is isolated by using separate processes.

Known Uses

- Currently, the separation of the components of the *Browser* in various processes, with different levels of access, is called as Modular Architecture [17]. This enables the separation of concerns in the browser, which gives greater stability, isolation, safety and speed.

- Google Chrome is based on the modular architecture, where each Renderer Process communicates with the Browser Kernel [3]. Internet Explorer, a proprietary browser, does not give much information about its structure or details of its implementation; [6] talks about Loosely-Coupled architecture [1] and its components, but without going into much detail. Firefox, meanwhile has two implementations: monoprocess and multiprocess/modular. Electrolysis is the name of the modular architecture being implemented, but has not yet been fully completed.

Related Patterns

- The Web Content Renderer pattern, which is under development, represents the subsystem hosted by a Sandbox that allows the parsing of a resource obtained through a request.

- The Browser Kernel pattern, also under development, represents the subsystem that represents the Web browser central component. This acts as a Reference Monitor [10] for all requests the Renderer does.

- The Sandbox is another name for the pattern Controlled Execution Domain [10].

Conclusions

A Web browser appears to be a medium complexity software for users and developers without security experience, but unfortunately this piece of software allows a variaty of attack vectors, to the user using it as well the system with which interacts. Therefore it is important to understand its structure and how it interacts with internal or/and external Stakeholders.

It is expected that in the future most *Web Browser* will take the form of a Modular Architecture. Therefore, it is important that developers know the internal processes of *browser* when developing a system that will interact with it. The Reference Architecture prepresented here, it is aimed at providing the basic knowledge of the components and interactions between *Web Browser* and external Provider for resources; as well as the threats that exist within it.

A part of our Reference Architecture has been built through the abstraction of found documentation, through the Browser Infrastructure pattern. We created our first architectural pattern for the infrastructure of *Web Browser*; to help others to understand holistically the components, interactions and relationships of this system. Furthermore it has been possible to characterize the Stakeholders and one of the most important use case. From what we have known, this is the second Reference Architecture for the *Browser* built.

The proposed work allows a better understanding of this system called Web Browser by using our partially Reference Architecture, this is also helpful to understand existing threats. Furthermore, as it is not subject to specific implementations, it is possible to generalize certain results in other browsers.

Future Work

Future work will be related to the creation of a Security Reference Architecture for the *Web Browser* using the same methodology presented here. Other patterns related to Browser Infrastructure pattern will be obtained in order to complete the AR already begun, such as the Web Content Renderer pattern and Browser Kernel. An example of the type of work to be carried out can be seen in [8] where this study carries out activities to build secure software and evaluate the safety levels of a system already built.

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. While at the same time these patterns will allow the construction of the Security Reference Architecture. 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. This is possible to perform under the same exercise already conducted in this work, looking for threats at each action for each use case of the Browser.

As for *Web Browsers*, attacks based on social engineering does not seem to decrease at any good time, because there is no current technology that can detect a 100% without no false positives the potential threats they can bring. Technologies such as CAMP (Content-Agnostic Malware Protection) appear to be part of the solution, but are still far from perfect.

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.