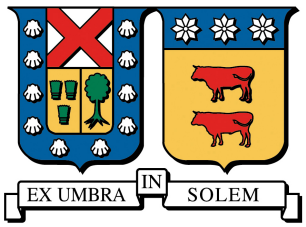
Universidad Técnica Federico Santa María
Departamento de Informática
Valparaíso, Chile



?????

Paulina Andrea Silva Ghio

Memoria para obtar al título de: Ingeniera Civil Informática

Profesor Guía: Raúl Monges
Profesor Correferente: Javier Cañas

25 de octubre de 2015

Resumen

El Web Browser es una de las aplicaciones más usadas - *killer app* - y también una de las primeras en aparecer en cuanto se creó el Internet (Década de los 90). Por lo mismo, su nivel de madurez con respecto a otros desarrollos es significativo y permite asegurar ciertos niveles de confianza cuando otros usan un Web Browser como cliente para sus Sistemas.

Actualmente muchos desarrollos de software crean sistemas que están conectados a la Internet, pues permite agregar funcionalidades al sistema y facilidades para sus *Stakeholders*. Esto lleva a depender de un cliente web, cómo un *Web Browser* que permita el acceso a los servicios, datos u operaciones que el sistema entrega. Sin embargo, la Internet influye en la superficie de ataque del nuevo sistema que se implementó, y lamentablemente tanto Stakeholders como muchos desarrolladores no están al tanto de los riesgos a los que se enfrentan.

Al tener sistemas que se interconectan con el Web Browser, tanto Stakeholder como Desarrolladores deben estar al tanto de los posibles riesgos que su proyecto enfrenta. La falta de educación de Seguridad en los Desarrolladores de un proyecto, la poca y dispersa documentación de los navegadores (así como su estandarización), podría llegar a ser un flanco débil en el desarrollo de grandes architecturas que dependen del Browser para realizar sus servicios. Una Arquitectura de Referencia del Web Browser, utilizando Patrones Arquitecturales, podría ser una base para el entendimiento de los mecanismos de seguridad y su Arquitectura, que interactua con un sistema Web mayor. Ésto mismo, entregaría una unificación de ideas y terminología, al dar una mirada holística, sin tener en cuenta detalles de implementación tanto del Browser como el sistema con el que interactua.

En esta Memoria presentada al Departamento de Informática (DI) de la UTFSM[1] Casa Central, se al incursionará en el ámbito de la seguridad del Web Browser, con el objetivo de obtener documentos formales que servirán como herramientas a personas que Desarrollen Software y hagan un fuerte uso del Navegador para las actividades del sistema desarrollado.

Abstract

The Web Browser is known as one of the most used applications - or *killer app* - and also was the first introduced when the Internet was created (1990). Which is why, it's significant maturity level is above in comparison with other developements and can assure a certain level of *trust* whenever it is used as a client with other systems.

[1]Universidad Técnica Federico Santa María

Índice general

Índice general II

Índice de figuras VI

Índice de cuadros VII

1. Introducción 1

1.1. Contexto General 1

1.2. El Problema: Desarrollo de Software y Seguridad 2

1.3. Motivación: ¿Por qué estudiar el Browser? 4

1.4. Contribuciones 5

1.5. Metodología 6

1.6. Estructura del Documento 7

2. Marco Teórico - Conceptual Framework 8

2.1. Definiciones Básicas 8

2.2. Browser 10

2.2.1. Arquitectura Cliente/Servidor 10

2.3. Procesos, Comunicación e Información de Estado 11

2.3.1. HTTP: Hypertext Transfer Protocol 11

2.3.2. SSL/TLS Encriptación en capa de Transporte 13

2.3.3. Speedy o Protocolo SPDY 14

2.4. Tecnologías usadas 15
2.4.1. Markup Languages 15
2.4.2. CSS: Cascading Style Sheets 15
2.4.3. DOM: Document Object Model 16
2.4.4. Javascript, VBScript y otros 17
2.4.5. Geolocalización 18
2.4.6. WebWorkers 18
2.5. SOP: Same Origin Policy 18
2.6. CORS: Cross-Origin Resource Sharing 20
2.7. Desafíos del Navegador 21
2.8. Arquitectura de Referencia o Reference Architecture (RA) 21
2.9. Desarrollo de Software Seguro y Diseño de Software Seguro 23
2.10. Patrones 24
2.11. Patrones de Mal Uso 24

3. Marco Teórico - (In) Seguridad en el Browser 26

3.1. Social Engineering 26
3.2. Ataques y Amenazas 27
3.2.1. Phishing 28
3.2.2. XSS - Cross-Site Scripting (DOM) 28
3.2.3. CSRF - Cross-Site Request Forgery 28
3.2.4. Session Fixation 28
3.3. Mecanismos de Defensa que se espera que el Host del Browser tenga previamente 28
3.4. Mecanismos de Defensa del Browser, para los ataques revisados 28
3.4.1. Sandboxing de procesos/componentes 28
3.4.2. Aislación del contenido web de los componentes del Browser . . 29
3.4.3. Blacklist y Whitelist de sitios web 29

3.4.4. Detección de Malware por medio de sistemas de Reputación . . 29

4. Arquitectura de Referencia del Web Browser 30

4.1. Casos de Uso del Browser 31

4.1.1. Stakeholders (actores) y Concerns de estos 31

4.1.2. Casos de Uso 31

4.2. Patrón Browser Infrastructure 33

4.2.1. Intent 33

4.2.2. Ejemplo 34

4.2.3. Contexto 34

4.2.4. Problema 34

4.2.5. Solución 35

4.2.6. Estructura 35

4.2.7. Dinámica 36

4.2.8. Implementación 38

4.2.9. Consecuencias 39

4.2.10. Ejemplo Resuelto 39

4.2.11. Usos Comúnes 40

4.2.12. Patrones Asociados 40

5. Patrones de Mal Uso 42

5.1. Identificando Amenazas y Patrones de Mal Uso 42

5.2. Identificando amenazas 42

5.3. Patrón de Mal Uso: Modificación de tráfico en el Web Browser 45

5.3.1. Intent 45

5.3.2. Contexto 45

5.3.3. Problema 46

5.3.4. Solución 47

5.3.5. Dinámica 47
5.3.6. Consecuencias 50
5.3.7. Contramedidas 51
5.3.8. Evidencia Forense 51
5.3.9. Patrones relacionados 51

Bibliografía **52**

Índice de figuras

1.1. Porcentaje de uso de Navegadores [1] 2

2.1. Arquitectura de DOM [2] 16

4.1. Diagrama de Caso de Uso del Web Browser 34

4.2. Componentes de alto nivel del Browser 36

4.3. Diagrama de Secuencia: Browser User Realiza request 41

5.1. Diagrama de Actividad para los casos de uso Realizar Request y Recibir Request 43

5.2. Diagrama de Clases para el patrón de Misuse 48

5.3. Diagrama de Secuencia para el Mal uso: Modificación de tráfico en el Web Browser 49

Índice de cuadros

5.1. Tabla con amenazas 44

Capítulo 1

Introducción

1.1. Contexto General

Entre 1989 y 1990, Tim Berners-Lee acuñó el concepto de *World Wide Web* y con ésto realizó la construcción del primer *Web Server*, *Web Browser* y las primeras páginas *Web*. Mucho antes que aparecieran los grandes sistemas que ahora conocemos, el *Web Browser* permitía navegar páginas estáticas y realizar una serie de acciones limitadas a las tecnologías de ese tiempo. En la actualidad el *browser* es la herramienta predilecta por todos, desde comprar tickets para una película, realizar reuniones por videoconferencia y muchas otras tareas que invitan a nuevas formas de interactuar y comunicar.

Durante la conocida *guerra de navegadores*, en la decada de los noventa, los browser tuvieron solo el objetivo de poder adquitir la mayor cantidad de usuarios posibles, entregando mejores funcionalidades que sus competidores. Debido a esto era habitual encontrar muchos parches que solucionaban problemas de seguridad, dada la cantidad de errores de programación y casi nula estructura del navegador, dado que no había una pronta preocupación por el diseño. Además la nula documentación que existía debido a la gran competitividad, muchas veces hacía que las extensiones hechas por *third-parties* creaban más agujeros de seguridad, que nuevas funcionalidades. El inicio del Software Libre u *Open Source*, cambió el escenario y las circunstancias pero aún así, existen navegadores propietarios que no exponen la arquitectura de sus aplicaciones.

En el último tiempo el mercado de los *Web Browser* ha crecido bastante (Figura 1.1), principalmente debido a la robustez que éstos poseen y a la cantidad de años que llevan desarrollándose en la industria del Desarrollo de Software. Los navegadores más conocidos son: Google Chrome o su versión Open Source Chromium, Firefox, Internet Explorer, Opera y Safari; siendo los primeros 3 el enfoque de este trabajo.

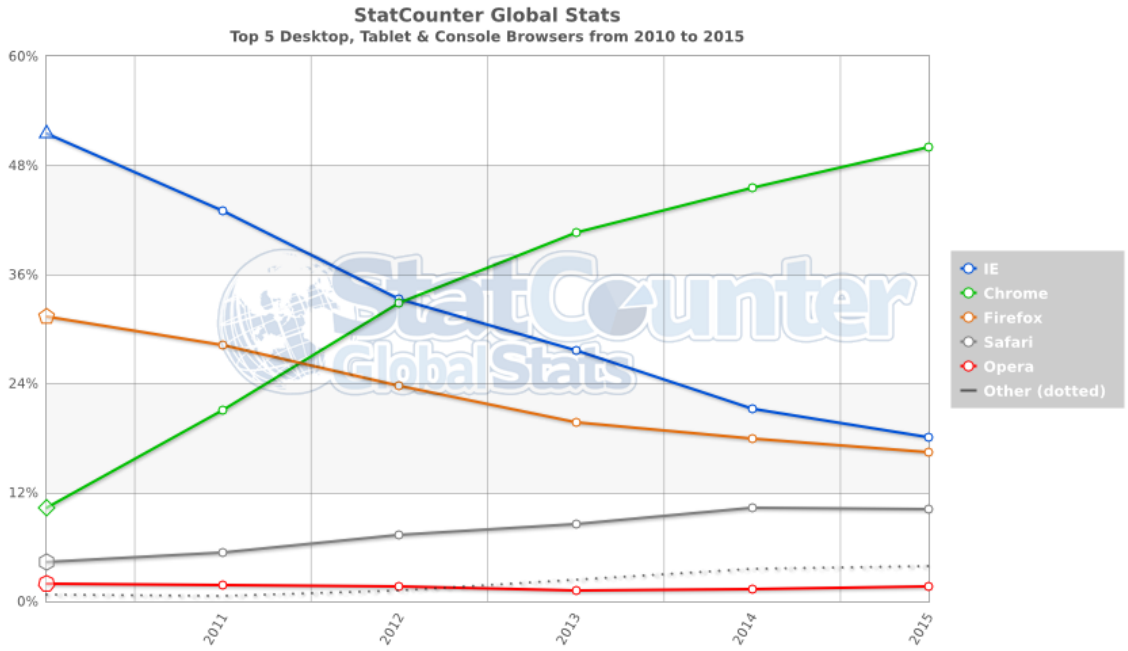


Figura 1.1: Porcentaje de uso de Navegadores [1]

La *Web 2.0* se inició con el uso intensivo de tecnologías como AJAX, y ésto ha permitido una nueva simbiosis entre el usuario, el Web Browser y el Servidor o Web Server que se comunican entre sí. El Navegador Web es una herramienta indispensable para todo tipo de tareas computacionales como comunicacionales, su existencia a penetrado completamente en las labores diarias de todos nosotros. En este mismo instante, la Web a evolucionado nuevamente obteniendo un nuevo nombre: *Web 3.0*, donde se realiza el uso de inteligencia artificial y sistemas de recomendación para generar nuevos tipos de contenido media para el usuario.

1.2. El Problema: Desarrollo de Software y Seguridad

Ningún Desarrollo de Software es igual al anterior. Por cada nuevo proyecto que surge es necesario ver qué tipo de proceso es el que se usará, qué personas serán parte del grupo de trabajo, qué condiciones económicas estará expuesto, qué *Stakeholder* están pendientes de que el Proyecto salga exitoso y un sin numero de variables, no menos importantes a considerar. Por lo tanto dependiendo de lo anterior, los sistemas podrían llegar a ser simples o muy complejos. En consecuencia, se hace necesario te-

ner ciertas metodologías que aseguren que se cumplan con todos los Requerimientos Funcionales como No-Funcionales del Sistema a construir. Sin embargo un problema que existe recurrentemente, es que la mayoría del Software construído contiene numerosos **defectos** y **errores**, generando así **vulnerabilidades** que son encontradas y explotadas por los atacantes, generandp un compromiso parcial o total del sistema [3]. Lo anterior sucede frecuentemente por que los sistemas no son desarrollados teniendo en cuenta la seguridad [4, 5, 6].

Muchas veces al desarrollar sistemas, se prefiere utilizar API's[1] de otros sistemas para poder incluir funcionalidades ya implementadas, fomentando así el Reuso de piezas de Software. Si bien lo anterior es una buena práctica, si el sistema no cuenta con las medidas de seguridad necesarias, estas piezas podrían ser causa de amenazas de seguridad que terminarían por corromper el sistema y en consecuencia podría causar una pérdida monetaria a los *Stakeholders*. Lo expuesto anteriormente ejemplifica perfectamente lo que tienen que lidiar los equipos de trabajo en proyectos de Desarrollo de Software, cuando dentro de sus preocupaciones la seguridad queda como un trabajo extra y no como parte del desarrollo completo.

Bien es sabido que un proyecto en producción que presente problemas que involucren a varias entidades, el costo asociado puede llegar a ser altísimo [7], sin olvidar que podría llegar a afectar la Confidencialidad, Integridad y Disponibilidad de los datos de los involucrados con el sistema [8]. Por esto mismo, es imperante que sean entendidos, desde el comienzo, los *concerns* de los *Stakeholders* y los Requerimientos de Seguridad asociados, y que además todos los involucrados los entiendan perfectamente. Según [6] la falta de preocupación en temas de seguridad en desarrollos de software no tiene una raíz principal, diversos factores como: la falta de estudios de seguridad en las mallas curriculares de las Universidades, pocos fondos para la investigación, la falta de iniciativa y precoupación desde la industria, el exceso de confianza de los desarrolladores, etc., son los causantes de las futuras vulneraciones en sistemas críticos.

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 como 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 [9], dado que la seguridad es una Propiedad Sistémica que necesita ser vista de manera holística y al inicio del proceso.

Un sistema cualquiera conectado al internet y accesado por un usuario a través de un Navegador, estará en algún momento de su vida útil bajo amenazas que el Desarrollador debe estar al tanto. Al conocer los peligros es posible comunicar a los *Stakeholders*, de los posibles peligros a los que se enfrentan, aún cuando en el

[1] Application Programming Interface

equipo de Desarrolladores no exista un experto en seguridad. Bajo esta misma lupa, una Arquitectura de Referencia permitiría comunicar efectivamente los componentes, interacciones y comportamientos existentes entre el Browser y el sistema con el que se comunica, de tal manera que sería posible entender los posibles problemas que el Browser puede llegar a generar.

1.3. Motivación: ¿Por qué estudiar el Browser?

Con la aparición de la *Web 2.0 y 3.0*, con el uso de *AJAX*, inteligencia artificial y sistemas de recomendación, permitieron nuevas formas de interacción entre usuarios y sistemas, lo que causó que el browser fuera usado extensivamente en los nuevos Desarrollos de Software dado que:

- Permite disminuir los costos de construir un programa Cliente (desde cero) para el usuario del sistema.
- Actualmente la Seguridad implementada en los *Web Browser* es bastante buena, dado que existen grandes compañias que se aseguran de ello (Google, Microsft, Mozilla entre las más conocidas).
- El *browser* es una herramienta indispensable. La mayoría de los sistemas que lo usan en la vida cotidiana son de tipo: *online banking*, declaración de impuestos, promoción de empresas o tiendas, compras, y mucho más.

Sin embargo los sistemas que dependen del uso del *Browser*, deben de tener en cuenta las posibles amenazas de seguridad a las que se enfrentarán por el solo hecho de usarlo. Para un proyecto de gran envergadura, sería un error no tener en consideración los posibles peligros que trae el uso del *Browser*, y es el deber de todo integrante del equipo de Desarrollo tener el conocimiento de la seguridad del Cliente Web. El entendimiento de la estructura subyacente del Web Browser podría asegurar que las personas que trabajen en el desarrollo, comprendan los *trade-off* al momento de diseñar un Software que necesite la colaboración del Navegador Web [10, 11, 12].

En [3, 6] mencionan que la cantidad de tiempo dedicado en temas de seguridad, por los estudiantes en áreas de la Información/Computación/Software es casi nula. Si bien existen diversos factores que pueden ser causa de este comportamientos en las universidades [6], la industria es un fuerte factor en la adopción de un cambio de mentalidad. Desarrollar un sistema crítico (o no) pero seguro, representa un gran desafío. No solo para los desarrolladores, si no también para los involucrados indirectamente, como los usuarios que navegan por el browser para hacer uso de éste sistema. Por lo mismo, es entendible que la seguridad sea un tema complicado de impartir, pero eso no significa que sea imposible. Nuestro trabajo tiene la intención de presentar un

trabajo conceptual, que permita entender más este atributo de calidad y que pueda ser ayudar a otros desarrollos ser más seguros.

Este trabajo tiene una motivación principal. Ésta es ayudar a quién lo necesite con el conocimiento necesario para entender el funcionamiento y construcción del Cliente, el Web Browser, los beneficios detrás de la Seguridad implementada en el Browser y de los peligros existentes de los que nos protegen. De esta manera se espera que alguien que lea este trabajo, tanto Estudiantes como Desarrolladores de Softwares, obtengan el conocimiento necesario al momento de trabajar junto con el Navegador Web al realizar un Desarrollo de Software que dependa de éste.

De nuestro conocimiento hasta el momento, no existe ninguna propuesta de Arquitectura de Referencia del Browser moderno, solo existe material desactualizado que no se adecua al paisaje actual. Creemos que una falta de conocimientos de seguridad con respecto al Browser, podría afectar de forma directa el desarrollo de aplicaciones que lo utilizan. Por esto mismo, una guía o compilado de información, semi-formal, que una los conceptos y componentes, como lo hace una Arquitectura de Referencia, podría enseñar a desarrolladores no expertos en seguridad, los peligros que existen. Nuestro trabajo explaya a la seguridad como una propiedad sistémica que debe ser tomada en cuenta desde el inicio del sistema [5, 13, 14, 9].

1.4. Contribuciones

Dada la falta de documentación para enseñar a desarrolladores no expertos en seguridad, acerca de los componentes, interacciones y comportamientos del Web Browser: El Objetivo General de esta Memoria es generar un cuerpo organizado de información sobre el Web Browser y su Seguridad, de tal manera que se pueda sistematizar, organizar y clasificar el conocimiento adquirido en un documento, con formato semi-formal, tanto para Profesionales como Estudiantes del área Informática que estén insertos en el área de Desarrollo de Software.

Este trabajo busca cumplir con los siguientes Objetivos Específicos:

- Comprender los conceptos relacionados al navegador web, sus componentes, interacciones o formas de comunicación, amenazas y ataques a los que puede estar sometido, como los también los mecanismos de defensa. Esto se realizará a través de un Estado del Arte sobre el Browser.
- Identificar actores, componentes, funciones, relaciones, requerimientos y restricciones del Navegador, para lograr abstraer una Arquitectura de Referencia (AR) a partir de documentación disponible en internet, blogs de desarrolladores, papers e iniciar un pequeño catálogo de Patrones de Mal Uso. Esto permitirá con-

densar el conocimiento obtenido en el punto anterior a través de documentos semi-formales, lo que permitirá generar una guía para comunicar los conceptos relevantes que pudieran afectar la relación existente entre un desarrollo de software y el navegador.

- Profundizar el conocimiento en ataques relacionados con métodos de Ingeniería Social.

Particularmente se ha escogido como metodología base la dada por el autor del libro [9]. Una Architectura de Referencia (AR) tiene como objetivo el mismo descrito en [11, 12, 15], éste es el ayudar a los *implementors* o desarrolladores del software, a entender los *trade-off* cuando se diseñan nuevos sistemas, y puede ayudar a los mantenedores de estos sistemas a entender el código *legacy* detrás los navegadores que trabajan a mano a mano. Además una Arquitectura de Referencia permite comparar las diferencias en decisiones de diseño del Navegador y así poder entender los cambios realizados a lo largo del Desarrollo de un sistema. Junto con lo anterior, Íla AR permitirá tener una visión holística del sistema y mostrará las decisiones de alto nivel para asegurar la Seguridad del sistema. Por otra parte, los Patrones de Mal Uso o Uso Indebido, permitirán enseñar y comunicar las posibles formas en que tal sistema puede ser usado inapropiadamente.

En este trabajo se presentará nuestra Arquitectura de Referencia y X Patrones de Uso Indebido, que usarán la AR contruída para mostrar los componentes y mensajes que una amenaza puede realizar, con tal de lograr un ataque en el Browser. El uso de patrones nos permitirá abstraer compontentes y comunicaciones entre dichos sistemas, al mismo tiempo que vislumbrará los mecanismos de seguridad implementados. Los patrones son herramientas de gran valor, que permiten generar el entendimiento de los aspectos funcionales y pueden complementar con otros patrones relacionados para alcanzar una arquitectura más entendible. Estos patrones serán presentados usando el template POSA [16] y UML, para así modelar las interacciones entre los diversos componentes de la arquitectura.

1.5. Metodología

Este trabajo se realizará de la siguiente forma:

1. Introducción de un *Framework conceptual* para entender los conceptos relacionados.
2. Contruir un Estado del Arte sobre el Browser, así como también de los posibles peligros a los que se enfrenta.

3. Identificar los conceptos, actores, componentes, interacciones y funciones, en relación al tema principal.

4. Construir patrones de arquitectura que definan los componentes y responsabilidades, con el objetivo final de ser unidos en una Arquitectura de Referencia.

5. Contruir patrones de Mal Uso/Uso Indebido por medio del punto anterior.

1.6. Estructura del Documento

El presente documento trata del trabajo de Memoria que se divide en las siguientes partes:

- En el capítulo **??**...
- Luego de tener un extenso conocimiento de lo que actualmente es conocido como **Web Browser**, el capítulo **??**

Capítulo 2

Marco Teórico - Conceptual Framework

2.1. Definiciones Básicas

Para empezar este estudio es necesario introducir ciertas nociones y lenguaje que se usarán durante todo el documento. Estos conceptos son usados en la Seguridad y Desarrollo de Software, y son extendibles para lo que se verá en este estudio.

- Seguridad - *Security*:
  Es una Propiedad que podría tener un sistema, donde asegura la protección de los recursos e información, en contra de ataques maliciosos desde fuentes externas como internas. La Seguridad también involucra controlar que el funcionamiento de un sistema sea como debería ser, y que nada externo o interno genere un error.

- Error - *Error*:
  Es una acción de caracter humano. Éste se genera cuando se tienen ciertas nociones equivocadas, que causan un Defecto en el Sistema o Código.

- Defecto - *Defect*:
  Es una caracterítica que se obtiene a nivel de Diseño, cuando una funcionalidad no hace lo que tiene que realmente hacer. Según la IEEE CSD o *Center for Secure Design* [17], un defecto puede ser subdividido en 2 partes: falla o **flaw** y **bug**, donde la primera tiene que ver con un error de **alto nivel**, mientras que un bug es un problema de implementación en el Software. Una falla es menos notoria que un bug, dado que ésta es de caracter abstracto, a nivel de diseño del Software.

- Falla - *Fail*:
  Es un estado en que el Software Implementado no funciona como debería de ser.

- Vulnerabilidades - *Vulnerability*:
  Es una debilidad inherente del sistema que permite a un atacante poder reducir el nivel de confianza de la información de un sistema. Una vulnerabilidad convina 3 elementos: un **defecto** en el sistema, un **atacante** tratando de acceder a ese defecto y la **capacidad** que tiene el atacante para llevarlo a cabo. Particularmente las vulnerabilidades más críticas son documentadas en la *Common Vulnerabilities and Exposures* (CVE) [18].

- Superficie de Ataque - *Attack Surface*:
  Es el conjunto de todas las posibles vulnerabilidades que un sistema puede tener, en un cierto momento, para una cierta versión del sistema, etc.

- Amenaza - *Threat*
  Es una acción/evento que se aprovecha de las vulnerabilidades del sistema, debilidades, para causar un daño, y que dependiendo del recurso al que afecte el daño puede o no ser reparable.

- Ataque - *Attack*
  Es el éxito de la amenaza en el aprovechamiento de la vulnerabilidad (explotación de ésta), de tal forma que genera una acción negativa en el sistema y favorable para el atacante.

- *Exploit*:
  Usar una pieza de software para poder llevar a cabo un ataque sobre un objetivo, intentando **explotar** la vulnerabilidad de éste. Este tipo de acción permite en consecuencia obtener control en el sistema computacional, en donde la vulnerabilidad permitió su acceso.

- Ingeniería Social - *Social Engineering*
  El acto de manipular a las personas de manera que realicen acciones o divulguen información confidencial. El termino aplica al acto de engañar con el propósito de juntar información, realizar un fraude, u obtener acceso a un sistema computacional. La definición anterior encontrada en Wikipedia es extendida por el autor del libro "The Social Engineer's Playbook" [19], donde agrega que además la Ingeniería Social involucra el hecho de manipular a una persona en realizar acciones que finalmente no son para beneficiar a la víctima. Un ataque de éste tipo también puede llegar a ser realizado tanto **cara a cara**, como de forma indirecta. Pero el autor del libro indica que siempre hay un **contacto** previo con la víctima.

- Confidencialidad - *Confidentiality*
  Característica o propiedad que debe mantener un sistema para que la informa-

ción privilegiada de alguna entidad que depende de tal sistema, no sea develada a nadie más que al que le pertenece la información.

- Integridad - *Integrity*
  Característica o propiedad que asegura que la información no será modificada/alterada nada más que por la entidad a quién le pertenece y con el previo consentimiento de éste.

- Disponibilidad - *Availability*
  Característica o propiedad que permite que la información esté disponible para quién lo necesite, en el momento que sea. La imposibilidad de obtener data en un cierto instante de tiempo, conlleva a la perdida de esta propiedad.

- *Phishing*
  Técnica de Ingeniería Social. Mediante el uso de correo elentrónico, links (url's), acortamiento de urls y otras herramientas, se busca que una victima visite un sitio o aprete un link de manera que se de la **autorización explicita** del usuario para descargar código malicioso o enviar datos a un servidor malicioso. El objetivo de esta técnica es poder obtener información valiosa de la victima o relizar algún daño en el cliente web.

- *Malware*
  Software creado para realizar acciones maliciosas en la data o sistema de un usuario. Puede ser instalado tanto de forma discreta como indiscreta, siendo la segunda opción causada a través de un ataque previo a cierta vulnerabilidad que permitió la instalación del malware, sin el consentimiento del usuario privilegiado del sistema.

- *Man-in-the-Middle*
  Ataque que causa una pérdida en la Confidencialidad de la información que es revelada. La causa de este ataque puede ser tanto:

  - Por técnicas de Ingeniería Social, entregano un certificado malicioso que el usuario acepta con o sin intención.
  - A través de vulnerabilidades del sistema que debieron ser explotadas antes para causar el ataque MiTM.

2.2. Browser

2.2.1. Arquitectura Cliente/Servidor

La web emplea lo que se conoce como una Arquitectura Cliente-Servidor, donde la comunicación entre ambas entidades se basa mediante mensajes de *request-response*

o solicitud-respuesta. Con el tiempo la forma en que se comunican estos programas a cambiado, desde iniciar solicitudes de forma secuencial e independiente, hasta solicitar asíncronamente varias peticiones. La evolución que ha tenido el cliente web ha permitido una mejor experiencia para el usuario, pero que conlleva ciertos riesgos que es necesario que el que usa el Browser sea consciente. De la misma manera que podemos afectar a un servidor a través de las solicitudes, las respuestas que el servidor envía al cliente pueden tener consecuencias graves [20].

2.3. Procesos, Comunicación e Información de Estado

2.3.1. HTTP: Hypertext Transfer Protocol

El Protocolo de la capa de Aplicación conocido como HTTP fue creado en los años 90 por el **World Wide Web Consortium** y la **Internet Engineering Task Force**, define una sintaxis y semántica que utilizarían los software basados en una arquitectura Web para comunicarse. El protocolo sigue un esquema de pregunta-respuesta o *request-response*, donde un cliente solicita un recurso que el servidor posee, y el servidor entrega una respuesta de acuerdo al recurso solicitado. La forma en que se localiza un recurso es mediante la dirección URL o *Uniform Resource Locator*

HTTP Headers

HTTP es la implementación de la capa de aplicacón del modelo OSI que sigue todo dispositivo que desea conectarse a la Internet. Los headers o cabeceras que utiliza este protocolo permiten configurar la comunicación entre un *Web Server* y un cliente web, en este caso con el Browser. Estos headers indican **dónde** debe ir el mensaje y **cómo** deben ser manejados los contenidos del mensaje. En cada petición o *request* del Navegador, éste debe especificarlos para que el servidor pueda enteder las peticiones; de la misma manera, el servidor enviará cabeceras que el cliente también debe entender. Algunos *headers* son necesarios y hasta obligatorios, para algunos servidores, y en otros da lo mismo como vayan.

- Content Security Policy: Es un mecanismo de defensa crea exclusivamente para la defensa de ataque de tipo XSS o *Cross-Site Scripting*. La misión de éste es definir bien la línea entre intrucciones y contenido, donde la primera se refiere a código que se debe ejecutar. Para que sea posible utilizar este mecanismo es necesario agregar al header del servidor, para la *request* del cliente, el header Content-Security-Policy o X-Content-Security-Policy, donde se indica la

localización de donde los scripts pueden ser obtenidos o *loaded* y además pone restricciones a estos mismos scripts.

- Secure Cookie Flag: El propósito de este header es de instruir al Browser de nunca mandar una *cookie* sobre un canal no seguro, solo debe ser realizado por HTTPS. Esta medida permite asegurar que una cookie tampoco será enviada por canales mixtos, donde al inicio de la comunicación HTTPS y luego vuelve a HTTP.
- HttpOnly Cookie Flag: Una opción para las *cookies* que permite inhabilitar el acceso al contenido de una cookie por medio de scripts. Esta opción originalmente fue pensada para evitar ataques XSS.
- X-Content-Type-Options: Un servidor que manda la directiva nosniff para este header, obligará al Browser a renderizar la página así como lo dice el header content-type. La idea de este header es poder limitar la ejecución del tipo objeto que pide el browser.
- Strict-Transport-Security: Obliga al navegador a que la comunicación con el servidor sea realizada por un tunel válido HTTPS, de manera que la comunicación sea completamente segura.
- X-Frame-Options: este header previene que se realize un *framing* de la página, es decir, esta opción evita que la página sea mostrada a través de un <iframe>. Este control permite especialmente mitigar ataques de *Clickjacking*, donde el usuario es engañado a través de lo que se muestra en la ventana del navegador.

Canales de comunicación en HTTP

Cuando se habla de HTTP usualmente ésto se relaciona con la comunicación que se lleva a cabo entre el cliente y servidor. Existen diversas formas para que ésto se lleve a cabo, las más conocidas son:

1. postMessage: Mecanismo de comunicación entre entre ventanas y frames, disponible en la API de HTML5, entre diversos dominios. El comando window.postMessage es usado para realizar llamadas entre diversos orígenes de forma segura. Si bien SOP normalmente denega este tipo de solicitud, si se hace de forma correcta es posible cominicarse con diversos orígenes por medio del uso de este comando.
2. XMLHttpRequest o XHR: [21] define una API que proporciona una guía de funcionalidad al cliente, para que éste pueda transferir datos del cliente al servidor. Dicho de otra manera, XHR es un objeto que permite la obtención de recursos

en la Internet. Soporta peticiones en HTTP o HTTPS, en general soporta toda actividad relacionada con un *HTTP request or response*, para los métodos definos.

3. WebSockets [22]: Es una tecnología nativa del Navegador que permite abrir un canal de comunicación interactivo, responsivo y *full-duplex* entre el cliente y el servidor. Éste comportamiento permite tener *event-driven actions* rigurosas sin necesidad explícita de sondear el servidor en todo momento. Websockets intenta reemplazar las tecnologías *Push* basada en AJAX.

4. WebRTC [23]: O mejor conocido como *Web Real-Time Communication*, es una API basada en la especificcación de la W3C, que utiliza las capacidades de Javascript y HTML5 (sin la utilización de plugins externos o internos) para transmitir audio, video y compartir archivos por medio de P2P. Ésta herramienta permite a los browsers comunicarse entre ellos a muy baja latencia y entrega un gran *bandwidth* para poder realizar comunicaciones media en tiempo real. Hasta el momento Google Chrome/Chromium y Firefox han implementado esta tecnología, con el objetivo de: mejorar la experiencia de usuario al no necesitar plugins para ser usada, y entregar seguridad dado que impone el uso de encriptación en los datos.

2.3.2. SSL/TLS Encriptación en capa de Transporte

Si bien existe una medida de seguridad en los headers que se implementa en la capa de Aplicación por medio de HTTP, ésto no impide que otros puedan ver qué contienen los paquetes. La confidencialidad, autenticidad y el no repudio de lo que se envía, es un aspecto relevante cuando se está trabajando con sistemas con información crítica y confidencial. SSL (Secure Socket Layer) y TLS (Transport Layer Security) tienen el objetivo de proveer un canal confiable y privado de todo lo que se envía entre dos aplicaciones que se comunican, es decir, una seguridad *end-to-end*. TSL es el resultado de la estandarización de SSL por la Internet Engineering Task Force (IETF). SSL/TLS trabaja debajo del protocolo HTTP usando certificados de clave pública que permiten:

- Resolver parcialmente el problema de la autenticación de un usuario, al establecer un canal seguro y encriptado mediante el uso de certificados digitales.
- Identificar que la información enviada por los dos *endpoints* sea solo de ellos dos, agregando una firma al final del paquete usando la clave privada de la entidad que envía.
- Asegurar que todo lo que se envía sea visto sólo por las entidades que crean el canal de comunicación, a través de la codificación de los paquetes con las claves

públicas de las entidades y su posterior decodicación con las respectivas claves privadas de cada uno.

El proceso que permite el inicio de una comunicación mediante SSL/TLS es:

1. Un usuario desea conectarse por el Browser a un Web Server.
2. Se inicia el proceso de *Handshake* entre el Browser y Servidor. Éstos dos se ponen de acuerdo en cómo se encriptará la comunicación (parámetros e información de los certificados) e intercambian una llave asimétrica.
3. El Navegador chequea la validez del certificado, ejemplo: revisa si está en una black list o está expirado o fue creado por una CA *Certificate Authority* confiable.
4. Si el servidor requiere un certificado por parte del cliente, el Browser le enviará el suyo. Esto permitirá tener una autenticación mutua entre las partes.
5. El Web Browser y el Servidor usan las llaves públicas del otro para poder acordar una clave simétrica, que es aquella que permitirá encriptar los mensajes. Sólo estas dos entidades conocerán tal clave.
6. El proceso de *handshake* termina y todo lo posterior se realiza encriptando los paquetes con la llave simétrica acordada por las partes.

Para que tanto SSL y TLS provean una conección segura, todos los componentes involucrados: cliente, servidor llaves y aplicación web deben ser seguros.

2.3.3. Speedy o Protocolo SPDY

Es un protocolo de red abierto desarrollado por Google en el 2009, para el transporte de contenido Web. A modo general utiliza técnicas de *multiplexing*, compresión y prioritización, sin embargo depende bastante de las condiciones del sitio web y su despliegue en la red. SPDY manipula el tráfico en el protocolo HTTP para disminuir el tiempo de carga de las páginas web, al mismo tiempo que cuida la seguridad de los datos. Este protocolo modifica la forma en que las peticiones y respuestas HTTP son enviadas a la internet (por el cable); SPDY es considerado una especie de tunel. Sin embargo, cuando la versión 2 de HTTP esté completa SPDY será deprecada. Implementaciones de este protocolo se dan en: Google Chrome/Chromium, Internet Explorer, Firefox, Safari, Opera y Amazon Silk.

2.4. Tecnologías usadas

2.4.1. Markup Languages

Un lenguaje de marcado sigue tradicionalmente un *Standard Generalized Markup Language*, de manera que entrega una semántica apropiada para representar o mostrar contenido, placeholders de aplicaciones y datos. Cada página mostrada por el navegador, sigue las instrucciones que el lenguaje de marcado le da al browser para mostrar el contenido. HTML y XML son los más conocidos en el mercado. Ambos lenguajes tienen sus especificaciones en la W3C o *World Wide Web Consortium*.

HTML: HyperText Markup Language

HTML [24], en especial la actual versión HTML5, es conocido por ser un *Simple Markup Language* o lenguage de marcado simple, usado principalmente para crear documentos de hypertextos que son posibles de portar desde una plataforma a otra, sin problemas de compatibilidad. Un documento HTML consiste de un árbol de elementos y texto, cada uno de esos elementos es denotado por un tag inicial y uno final; estos tags pueden ir anidados y la idea es no se superponen entre ellos. Un HTML User Agent o Browser consume el HTML y lo parsea para crear un árbol DOM, que es la representación en memoria del documento HTML. Una característica importante de este lenguaje de marcado es su flexibilidad ante los errores, esto es que en alguna ocasiones el programador perfectamente podría sobrarle un signo y HTML no le daría mayor importancia mientras no afecte a la estructura global de la página. Normalmente esta característica es aprovechada por los atacantes para insertar nuevos elementos html que ejecuten scripts que afectarían al navegador.

XML: eXtensible Markup Language

Este lenguaje de marcado tiene una estrecha relación con HTML, pero a diferencia de este último tiene una sintáxis y semántica más rígida ya que sigue al pie de la letra un lenguaje libre de contexto. Este tipo de lenguaje es ideal para el transporte de data entre *web Services* o interacciones **RPC**, dado que no hay forma de como malinterpretar la data.

2.4.2. CSS: Cascading Style Sheets

Es un lenguaje usado junto a HTML o XML para definir la capa de presentación de las páginas web que el navegar renderiza al usuario. La W3C se encarga de la

especificación de las hojas de estilos para que los browser sean capaces de interpretar bajo estándares y aseguren ciertos niveles de calidad. Una hoja de estilo se compone de una lista de reglas. Cada regla o conjunto de reglas consiste en uno o más selectores y un bloque de declaración, más los estilos a aplicar para los elementos del documento que cumplan con el selector que les precede.

2.4.3. DOM: Document Object Model

Es una *API* independiente del lenguage y multiplataforma para HTML válido y bien formado, que define la estructura lógica de un documento que permite ser accedido y manipulado. DOM es una especificación que permite a programas Javascript modificar la estructura del contenido de una página dinamicamente. Esto permite que una página pueda cambiar sin la necesidad de realizar nuevas peticiones al servidor y sin la interacción del usuario. Posteriormente la *W3C* [2] formó el *DOM Working Group* y con ello se creó la especificación a través de la colaboración de muchas empresas y expertos. La arquitectura de esta *API* se presenta en la Figura 2.1, donde el *Core Module* es donde están las interfaces que deben ser implementadas por todas las implementaciones conformes de DOM. Una implementación de DOM puede ser construida por uno o más módulos dependiendo del host, ejemplo de esto: la implementación de DOM en un servidor, donde no es necesaria la implementación de los módulos que manejen los triggers de eventos del mouse.

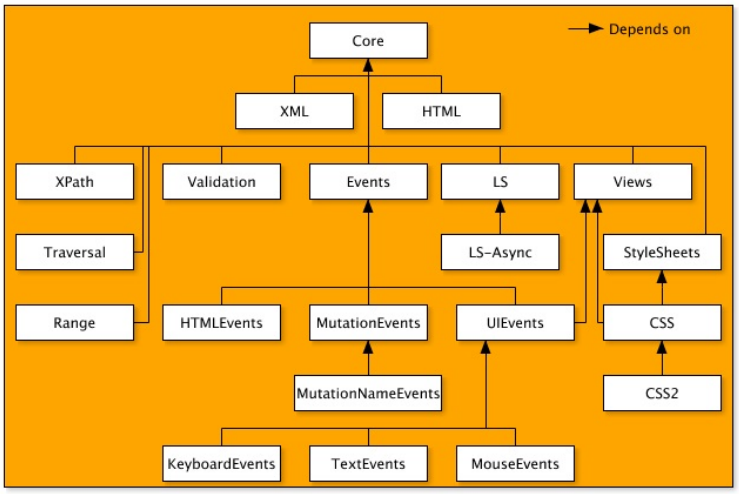


Figura 2.1: Arquitectura de DOM [2]

La interfaz de *DOM* fue definida por el **OMG IDL** y fue construida para ser usada en una gran variedad de ambientes y aplicaciones. El documento parseado por DOM se transforma en un gran objeto, tal modelo captura la estructura del documento y

el comportamiento de éste, además de otros objetos de lo que puede estar compuesto y las relaciones entre ellos. Cada uno de los nodos representa un elemento parseado del documento, el cuál posee una cierta funcionalidad e identidad. La estructura de árbol del DOM construido puede llegar a ser gigantezca, y almacena más de un árbol por cada documento que parsea.

2.4.4. Javascript, VBScript y otros

Ambos son lenguajes de scripting orientados a objetos. Javascript fue desarrollado por Netscape mientras que VBScript fue desarrollado por Microsft para Internet Explorer, los dos siguen el estandar del lenguage de scripting **ECMAScript**. Dado que VBScript no era usado por muchos y no tenía soporte para otros navegadores más que Internet Explorer, Microsoft decidió abandonarlo.

Muchos piensan que JavaScript es un lenguage interpretado, pero es más que eso. Javascript es un lenguage de **scripting dinámico** (por tanto no tipificado) que soporta la construcción de objetos basados en **prototipos**. Esto quiere decir que a diferencia de un lenguage de programación orientado a objetos como Java, un lenguage orientado a prototipos no hace la distinción entre clases y objetos (clase instanciada), son simplemente objetos. Y cómo tal al ser construido con sus propidades iniciales, es posible poder agregar o remover propiedades y métodos de forma dinámica (durante el runtime) tanto a un objeto como a la clase.

Javascript puede funcionar tanto como un lenguage de programación procedural o como uno orientado a objetos. Firefox usa una implementación en C de Javascript llamada *Spider Monkey*, Google Chrome/Chromium tiene un motor de JavaScript llamado *V8* e Internet Explore no usa realmente JavaScript si no que *JScript* (hace lo mismo que las otras implementaciones solo que difiere en el sistema operativo que utiliza) que en este caso se llama *Chakra*.

Si bien es posible comprender que JavaScript posee increíbles posibilidades para la creación de *RIA* (Rich Internet Applications), en [25] se muestra que puede llegar a ser un fracaso si es que no se toman en cuenta ciertas vulnerabilidades inherentes al lenguage. Estas vulnerabilidades que pueden llegar a ser criticas, a menudo permiten a un comunicante comprometer completamente a la otra parte. La misma naturaleza de JavaScript que permite la modificación en runtime de los objetos, puede llegar a ser aprovechada de esta situación; en la cita toma por ejemplo la comunicación entre los elementos de un *Mashup*.

2.4.5. Geolocalización

Cada Browser posee una API que permite obtener la data de la localización del host donde el browser está alojado. Ésta es obtenida ya sea del GPS, si es un dispositivo móvil, como de la triangulación de la señal del celular, localización de IP del movil o *access point*.

2.4.6. WebWorkers

Ésta tecnología permite la creación de *threads* en el browser para separar las tareas de éste, dejando algunas en el *background* para incrementar el rendimiento total de la carga de las páginas web. La API permite que el autor de una aplicación web, ejecute *trabajadores* que corren scripts en paralelo, la coordinación entre estos se logra a través del paso de mensajes. Existen 2 tipos: una que es compartida por todo aquello de un mismo **Origen** y otra que se comunica hacia atrás a la función que la creó. Esta API entrega al desarrollador más flexibilidad, pero que sin duda los atacantes también aprovechan bastante.

2.5. SOP: Same Origin Policy

Es un principio de seguridad implementado (hoy en día) por cada browser existente, su principal objetivo es restringuir las formas de comunicación entre una ventana y un servidor web. **Same Origin Policy** o **SOP** es un acuerdo entre varios manufacturadores de navegadores web cómo Microsoft, Apple, Google y Mozilla (entre los más importantes), en donde se definió una estandarización de cómo limitar las funcionalidades del código de scripting ejecutado en el browser del usuario.

Este importante concepto nace a partir del Modelo de Seguridad detrás de una Aplicación Web, al mismo tiempo que es el mecanismo más básico que el Browser tiene para protegerse de las amenazas que aparecen en el día a día, haciendo un poco más complicado el trabajo de crear un *exploit*. **SOP** define lo que es un **Origen**, compuesto por el **esquema**, el **host/dominio** y **puerto** de la URL. Esta política permite que un Web Browser aisle los distintos recursos obtenidos por las páginas web y que solo permita la ejecución de *Scripts* que pertenezcan a un mismo **Origen**. Inicialmente fue definido solo para recursos externos, pero fue extendido para incluir otros tipos de orígenes, esto incluye el acceso local a los archivos con el *scheme* **file://** o recursos relacionados al Browser con **chrome://**.

SOP puede distinguir entre la información que envía y recibe el Web Browser, y solo se aplicará la política a los elementos externos que se soliciten dentro de una

página web (recepción de la información). Esta imposibilidad de recibir información de un **Origen** diferente al del recurso actual, permite disminuir la superficie de ataque (*Attack Surface*) y la posibilidad de explotar alguna vulnerabilidad en el sistema donde reside el Browser. Sin embargo, **SOP** no pone ninguna restricción sobre la información que el usuario puede enviar hacia otros. Sin **SOP** cualquier sitio podría accesar a la información confidencial de un usuario o de cualquier otro sitio. Por tanto es sencillo entender la razón de la existencia de **SOP**, se desea proteger la información del usuario, sus cookies, token de autenticación, etc. de las amenazas existentes en la Internet.

En [26] menciona que no existe una sola forma de **SOP**, si no que es una serie de mecanismos que superficialmente que se parecen, pero al mismo tiempo poseen diferencias:

- **SOP** para acceso al **Document Object Model**: se dará permiso de modificar el **DOM** y sus propiedades solamente aquellos scripts que tienen el mismo dominio, puerto (para todos los browsers excepto Internet Explorer) y protocolo. Visto de otro modo, el mecanismo entrega una especie de Sandboxing para el contenido potencialmente peligroso y no confiable. Sin embargo éste no es suficiente, pues posee varias desventajas: el dominio es posible de cambiar a la conveniencia del atacante, limita las acciones a los desarrolladores lo que se traduce en que éstos tengan que buscar bugs que permitan liberarse de estas restricciones lo que incita a atacantes a aprovecharse de ésto.
- **SOP** para el objeto XmlHttpPRequest: para diferentes tipos de peticiones (GET, POST y otros) existen condiciones y suposiciones que hacen que se tome o no en cuenta el *request* del cliente, además del uso de una *whitelist* de las formas en que el header de la petición puede salir del browser.
- **SOP** para *cookies*: restringuiendo el uso de acuerdo su dominio, *path*, tiempo de uso, modificando o eliminado las cookies, e incluso protegiendo las cookies usando el *keyword: secure*. Sin embargo, desde su implementación las cookies han generado bastante problemas de seguridad.
- Y otros como: SOP para Flash, donde usa políticas para realizar peticiones fuera del dominio através de un archivo **crossdomain.xml**, **SOP** para Java y **SOP** para Silverlight, parecido al de Flash solo qe utiliza otros elementos.

Tanto para los atacantes como desarrolladores de Software, SOP puede llegar a ser bastante molesto. Para el primero, la respuesta es obvia, pero para el segundo está el problema de ¿cómo poder aislar los componentes no confiables o parcialmente confiables, mientras que al mismo tiempo se pueda tener una comunicación entre ellos de forma segura? Ejemplo de esto son los Mashup [27], que permiten juntar contenido de terceros en una misma página por medio de frames, etc.

Existen excepciones que permiten evitar el uso de SOP, pero como es de esperar esta vía puede ser mal usada por los atacantes en contra del usuario y de la Aplicación Web. Dentro de las excepciones están los elementos en HTML <script>, <img>, <iframe> y otros, que si bien permiten la comunicación entre diferentes orígenes, un mal uso de este puede causar grandes estragos, desde la eliminación de registros en una base de datos hasta la propagación de un gusano o virus.

Queda decir que si bien SOP entrega una capa de seguridad al usuario y a la Aplicación Web, contra cierto tipo de ataques (muchas veces del tipo de ataques de principiantes), esto no es suficiente. Es responsabilidad del desarrollador de Software poseer las herramientas necesarias para asegurar la confidencialidad e integridad del sistema a través de otros métodos de seguridad.

2.6. CORS: Cross-Origin Resource Sharing

Cómo lo define su nombre es un mecanismo (especificación) que permite al cliente realizar request entre sitios de diverso *Origen*, ignorando el **SOP**. *CORS* define una forma en que el Browser y el Servidor Web puedan interactuar para determinar si permitir o no el request a otro origen. Un Browser utiliza SOP para restringuir los request de la red y prevenir al cliente de una Aplicación Web ejecutar código que se encuentra en un origen distinto, además de limitar los request HTTP no seguros que podrían tratar de generar un daño. CORS extiende el modelo que el Browser maneja e incluye:

- Un header en la respuesta/response del servidor solicitado llamado *Access-Control-Allow-Origin*, donde se debe escribir el origen que tendrá acceso a los recursos solicitados al servidor. Si el valor de la respuesta del servidor coincide con el *origen* de quién lo solicitó, se podrá realizar el uso del recurso en el navegador, de lo contrario se generará un error.
- Otro header llamado *Origin* pero esta vez en el request de la solicitud, para permitir al Servidor hacer cumplir las limitaciones en las peticiones de distinto origen.
- En algunos casos un browser deberá agregar el header *Access-Control-Allow-Methods*, ya que el servidor no responderá de vuelta si no es así. Esto permite limitar la superficie de ataque en el servidor.

Existen ciertos métodos en HTTP que necesitan realizar un *pre-vuelo* antes de ser ejecutados, si la response del servidor es afirmativa luego se enviará el request original con el método que se debió confirmar su utilización. Para el caso de los métodos GET

y POST, los más usados, este pre-vuelo no es necesario y se puede enviar el request inmediatamente.

La gran diferencia de CORS con cualquier otro método de que permita hacer request hacia un origen distinto, es que el Browser por default no enviará ningún tipo de información que permita identificar al user. De esta manera se puede disminuir considerablemente las amenazas en la confidencialidad, pues el atacante no podrá hacerse pasar por un usuario del que no tiene información. Casi todos los navegadores web, a diferencia de Internet Explore [28], realizan sus solicitudes a servidores de diverso origen por medio de la interfaz *XmlHttpRequest*, en el caso de Internet Explorer esta se llama *XDomainRequest*.

2.7. Desafíos del Navegador

- Navegación a todo tipo de contenidos/compatibilidad: sin importar el esquema de la página web, el navegador debe ser capaz de presentar todo tipo de contenido al usuario. [29] asegura que los usuarios demandan compatibilidad, por que el browser es sólo útil mientras pueda mostrar las páginas.
- Navegación personalizada: el browser debe ser capaz de entregar la información a la Web Aplicación/sistema, para que identifique al usuario por detrás, de manera que la navegación sea personalizada.
- Navegación sin inconvenientes: el navegador debe ser capaz de aislar los errores que se presenten en algunas páginas, de tal manera que no molesten a las otras páginas web que se ven al mismo tiempo.
- Seguridad: Los datos de los usuarios y el host donde se mantiene el Browser no deben ser expuestos a terceras partes.

2.8. Arquitectura de Referencia o Reference Architecture (RA)

Una arquitectura de Referencia, de acuerdo a la *Open Security Architecture* o OSA[30], es considerado un elemento que describe un **estado de ser** y debe representar aceptadas buenas practicas. En [15] se explica que una RA es una arquitectura de software genérica y estandarizada, para un dominio particular e independiente de la plataforma o detalles de implementación. En ésta especifica la decomposición del sistema en subsistemas, las interacciones entre éstas partes y la distribución de funcionalidad entre ellas [31].

Actualmente no hay un consenso de como definir una AR o lo que debería contener [32], [15] describe un ejemplo e indica como debería de ser ésta con los siguientes elementos:

- Describir los Stakeholders que interactuan con el sistema y que poseen *concerns* de éste.
- Generar *views* usando UML y teniendo en cuenta un proceso *Rational Unified Process*: crear casos de uso, modelos de análisis y diseño, modelo de despliegue e implementación.
- Patrones de Arquitectura.
- Atributos de calidad deseables que el sistema debe garantizar. Es importante solo destacar aquellos realmente necesarios, dado que un sistema sobrecargado con ellos tampoco es conveniente.

Las ventajas y usos que se obtienen al construir una RA son:

- Comprender la estructura subyacente de un Web Browser y las interacciones que tendrá con otros sistemas.
- Proveer una base tecnológica modular y flexible. Al tener los subsistemas compartimentalizados es posible quitar y sacar piezas, que poseen interfaces similares, y de esa manera reusar lo otro sin tener que construir un sistema nuevo.
- Entrega una base para el desarrollo de otros Navegadores Web, sin explicar detalles de implementación.

En este trabajo el enfoque estará en el primer punto, donde se quiere entender las interacciones entre un desarrollo de Software y la utilización de las funcionalidades del Navegador. Dado que parte de la investigación es obtener Patrones de Mal Uso o Uso Indebido del Navegador Web, es primordial concebir una Arquitectura de Referencia que permita encontrar donde es posible aplicar Patrones de Seguridad para poder mitigar los malos usos del Browser [33].

Una RA es una herramienta que permite facilitar el entendimiento de sistemas complejos y su apropiada construcción a sistemas reales. Si bien una RA es usada principalmente para capturar los *concerns* de los *Stakeholders* al comienzo de un Desarrollo de Software, también puede ser usada para educar al realizar la unión de ideas y terminologías usadas por diversos sistemas que se asemejen. Una Arquitectura de Referencia debe ser en lo posible descrita de la forma más abstracta posible, pues su función guiar la construcción de arquitecturas concretas, sin tener en cuenta detalles de las tecnologías usadas.

[34] menciona que existe una falta de procedimientos para diseñar sistemáticamente una Arquitectura de Referencia, que sea al mismo tiempo fundamentada empíricamente. El mismo trabajo explica que mientras un Arquitecto de Sistema y un Experto de Dominio trabajen juntos, es posible diseñar AR ya sea desde *cero* o basado en artefactos arquitecturales ya existentes. Para una AR no construída desde el comienzo, la evaluación es menos crítica dado que la AR utiliza conceptos arquitecturales ya comprobados por expertos. Por lo tanto, la validación de ésta puede derivarse desde arquitecturas ya construídas, en este caso a partir de los browser que se encuentran en el mercado.

Para describir la Arquitectura de Referencia nos hemos basado en los trabajos [35, 33], usando patrones para la contrucción de la AR. Así como indica [31], es posible usar patrones arquitecturales para diseñar una Arquitectura de Referencia, con tal de obtener atributos de calidad deseados.

2.9. Desarrollo de Software Seguro y Diseño de Software Seguro

La filosofía detrás de *Secure Software Developmet* es que detrás de cada etapa de desarrollo del software, se tengan en cuenta los prinicipios de Seguridad: Confidencialidad, Integridad, Disponibilidad y Auditoría. Para cumplir este cometido es que se deben llegar a políticas y reglas que aseguren la Seguridad como una propiedad sistémica.

Varias comunidades tienen diferentes enfoques y técnicas de cómo asegurar la Seguridad en los sistemas, muchas pueden incluso tener similitudes y hasta trabajar juntas. En este trabajo, el enfoque tomado es aquél que busca entregar la propiedad de seguridad a través del entendimiento de un sistema a un alto nivel, identificando las amenazas durante la elicitación de requerimientos, de manera que se pueda extraer las posibles amenazas que podrían existir y utilizando elementos de diseño para hacer cumplir los principios de seguridad necesarios por el sistema; este enfoque es el que se dedica la comunidad de *Secure Software Design*.

Fernandez [9] sostiene que para construir un sistema seguro es necesario realizarlo de manera sistemática de tal manera que la seguridad sea parte del integral de cada una de las etapas del Desarrollo de Software - de inicio a fin. El enfoque que propone es ingenieril y por tanto es aplicable incluso para sistemas *legacy*, donde es posible hacer ingeniería inversa para comprobar si existen o no los requerimiento de seguridad implementados, de manera que permite generar un estudio con la intención de comparar y mejorar nuevos sistemas. En su libro [9] presenta una completa metodología para construir sistemas seguros a partir del Diseño Orientado a Objetos, UML

y patrones, a los cuales nombra como **Security Patterns**.

Como parte de la metodología propuesta, se plantea que para diseñar primero se deben entender las posibles amenazas a las que está expuesto el sistema. La identificación de Amenazas [14, 13] es la primera tarea que presenta la metodología, que considera las actividades en cada caso de uso del sistema.

2.10. Patrones

Los Patrones encapsulan soluciones recurrentes a problemas y definen una forma de expresar los requerimientos y soluciones de una forma concisa, al mismo tiempo que proveen de un vocabulario común entre los diseñadores [16]. Un patrón encarna el conocimiento y experiencia de desarrolladores de software que puede ser reusado posteriormente en nuevas aplicaciones [5, 9]. Los Patrones expresan las relaciones entre un contexto, un problema y una solución. Para un contexto dado, el patrón puede ser adaptado para encajar en diversas situaciones. La construcción de Patrones de Seguridad parte de la premisa anterior, éste permite construir sistemas seguros a través del uso de Patrones adaptados a las necesidades del sistema y preocupaciones de los *Stakeholders*. Por otra parte, una Arquitectura puede ser descrita a través de Patrones, permitiendo que haya un mejor entendimiento al momento de proveer con guías de diseño y análisis a desarrolladores.

Los patrones describen diseños recurrentes en un mediano nivel de abstraction y es poco probable que existan solos, es decir, existen en conjunto a otros patrones. Un patrón puede proveer una solución usando diagramas en UML, de manera que describen de forma precisa al sistema.

La Arquitectura de Referencia a confeccionar será realizada por medio de patrones y éstos serán descritos con el template creado por [16], llamado POSA, que contiene las siguientes secciones para describir un patrón: *Intent*, Contexto, Problema, Solución, Implementación, Usos comúnes, Consecuencias y Patrones relacionados.

2.11. Patrones de Mal Uso

Para diseñar sistemas seguros, se es necesario identificar las posibles amenazas que un sistema puede sufrir. Papers como [13, 36, 14, 9] describen el desarrollo de una metodología completa para encontrar amenazas, a través del análisis de actividaddes de los casos de uso del sistema buscando como podría un atacante interno o externo socavar las bases de esas actividades. Es importante no confundir *Attack Patterns* con *Misuse Pattern*, pués claramente en [37, 9] dejan explícito que un *Attack Patttern* es

una acción que lleva a un mal uso o *misuse*, o acciones **específicas** que toman ventaja de las vulnerabilidades de un sistema, como por ejemplo un *buffer overflow*. A partir de los trabajos [36, 38, 39] se hace la unión de los conceptos de *Attack Patttern* para dar forma a la definición de *Misuse Pattern* [37, 40, 41, 42, 43, 44, 45, 46]:

> Un patrón de mal uso o *Misuse Pattern* describe, desde el punto de vista del atacante, cómo un tipo de ataque es realizado (qué unidades usa y cómo), analiza las maneras de detener el ataque através de la enumeración de posibles Patrones de Seguridad que pueden ser aplicados, y describe cómo rastrear un ataque una vez que ha ocurrido por medio de una recolección y observación apropiada de datos forenses.

Sin embargo, cuando un sistema ya está diseño y construido, como es el caso del Web Browser, lo que va a importar es saber **cómo** los componentes del sistema, pueden ser usados por el atacante para alcanzar sus objetivos. Un *Misuse Pattern* o **Patrón de Mal Uso** describe, desde el punto de vista del atacante, cómo un tipo de ataque es realizado, indicando **qué** componentes usa y **cómo**. Además analiza las formas de detener el ataque a través de un listado de posibles *Security Patterns* o **Patrones de Seguridad** que pueden ser aplicados para esa situación, y describe cómo poder seguir el rastro de un ataque una vez que ha sido realizado con éxito en el sistema, a través de data forense. Además describe un contexto en dónde puede ocurrir el ataque.

Un catálogo de *Misuse Patterns* podría ser de gran valor en el Desarrollo de Sistemas que interactúan con el Navegador, pués provee a desarrolladores un medio para evaluar los diseños de sus sistemas, al analizar las posibles amenazas del Browser que pusieran afectar al software que está siendo construído.

Capítulo 3

Marco Teórico - (In) Seguridad en el Browser

En esta sección se presentan los posibles ataques que un Browser puede sufrir y que directamente podrían afectar al sistema con el que se comunica. Principalmente ahondaremos en los ataques en el Browser relacionados a las técnicas de Ingeniería Social [19]. El escenario actual de los ataques en el browser ha cambiado bastante, si es comparado a aquellos de la decada de los noventa. Cada día los Browsers son más robustos y difíciles de explotar, y por lo mismo los ataques de tipo *drive-by downloads* o los basados en ejecución de código para vulnerar el sistema, cada vez son menores. Una nueva forma de ataque ha emergido y es bastante fácil de lograrlo, pues se basa en el engaño del usuario a realizar lo que el atacante desea. Una vez el usuario es engañado, el atacante puede lograr un control total tanto del Browser o del Host, sin haber tenido que vulnerar el sistema [47] que aloja al Browser. Desarrollos de sistemas críticos que interactuan a diario con diferentes usuarios en la red, deberían de ser los más preocupados de estos ataques pues atentan contra la confidencialidad, integridad y disponibilidad de los datos, tanto del usuario (personales) como los de los *Stakeholders* involucrados.

3.1. Social Engineering

[19] define este tipo de acción como: El acto de manipular una persona para realizar acciones que no son parte de los mejores intereses del *blanco o víctima* (la misma persona/organización/etc u otra entidad). Un ataque de éste tipo puede darse de diversas maneras, no dejando la posibilidad de un encuentro físico o digital con el que realiza el engaño. Un ataque basado en ingeniería social, es uno que se aprovecha del comportamiento humano y la confianza de la víctima. En el contexto del Web

Browser, el usuario engañado es la primera y última línea de defensa contra este tipo de ataques, pues un abuso en la confianza del usuario podría abrir las puertas al Host del Browser, logrando un daño tanto del usuario como con los sistemas externos con los que interactúa.

3.2. Ataques y Amenazas

Esta sección incluye todos los ataques y amenazas posibles de realizar en un Browser y que podrían afectar tanto directa como indirectamente a un sistema externo. Acá no incluiremos ataques en donde el el Host ya ha sido vulnerado con anticipación, o aquellos que puedan correr software con los privilegios de un usuario del sistema Host, es decir, aquellos donde el Host ya ha sido controlado directamente. En el caso anterior, los Browsers ya nada pueden hacer para detener un ataque de esa magnitud.

En el Top Ten [48] de la OWASP (Open Web Application Security Project) - los diez riesgos de seguridad más importantes en Aplicaciones Web - se puede distinguir en el año 2013 los riesgos directamente relacionados a amenazas de seguridad en el Browser. Algunos como: Injección (A1), Manejo de sesiones y autenticación roto (A2), XSS (A3), CSRF (A8) y uso de componentes con vulnerabilidades conocidas (A9), son los riesgos que las organizaciones podrían sufrir en sus sistemas cuando se realizan ciertos ataques en el Browser.

En trabajos [29, 49] se puede apreciar los Threat Model para los Navegadores Chromium y Firefox respectivamente. Es posible observar que existen ataques que pueden generar secuelas en otros sistemas, si es que el Navegador es afectado en primera instancia. Algunas amenazas existentes relacionadas con los riesgos mencionados en el párrafo anterior son:

1. Compromiso de los componentes del Navegador (plugins incluídos) que poseen privilegios de usuario.
2. Compromiso del Host/Sistema.
3. Robo de datos en una red.
4. Compromiso de páginas web de origenes distintos.
5. Comprimiso de la data de páginas web de orígenes distintos.
6. Fijación de sesión o robo de ésta
7. Compromiso de los canales de comunicación del Browser.

Una lista (parcial) de ataques asociados a las amenazas anteriores son:

3.2.1. Phishing

Instalación de Malware o Extensiones malignas

3.2.2. XSS - Cross-Site Scripting (DOM)

3.2.3. CSRF - Cross-Site Request Forgery

3.2.4. Session Fixation

3.3. Mecanismos de Defensa que se espera que el Host del Browser tenga previamente

Antes de revisar los mecanismo que un Navegador nos ofrece, es importante

3.4. Mecanismos de Defensa del Browser, para los ataques revisados

3.4.1. Sandboxing de procesos/componentes

La idea es encapsular el área de mayor probabilidad de ataque en un espacio aislado, minimizando la superficie de ataque de un software. Sandboxing no es una técnica tan nueva, han existido sistemas que ya lo han incorporado. Ésta protección puede ser aplicada dependiendo del diseño del software, algunos ocupan Sandbox a nivel del sistema operativo como otros que ocupan al nivel del *engine* de Javascript. En el caso especial del Browser, esta técnica es construida en el nivel más alto posible para un programa de usuario, lo que permite la separación de privilegios entregados por el sistema operativo al browser y los subprocesos que corren dentro de éste. El atacante que se enfrente a un browser que tenga este mecanismo de defensa, tendrá que realizar primero un *bypass* encontrando una vulnerabilidad en el sandboxing del browser. Existen diferentes técnicas para Sandboxing, todo depende del diseño del Browser.

3.4.2. Aislación del contenido web de los componentes del Browser

3.4.3. Blacklist y Whitelist de sitios web

3.4.4. Detección de Malware por medio de sistemas de Reputación

Capítulo 4

Arquitectura de Referencia del Web Browser

Para poder entender algún Browser se buscó en específico trabajos, papers, documentación gris a través de buscadores web como: IEEE Xplore, ACM digital Libray, Scopus y otros más, con tal de encontrar documentos formales sobre sus arquitecturas. Lo que más se encontró fueron blogs de desarrollo y White papers sobre Google Chrome, sin embargo llama la atención la escases en literatura sobre el tema.

En vista a la poca, casi nula, documentación formal sobre el Navegador Web, la necesidad de hacer una AR para entender cómo la arquitectura de este sistema puede relacionarse con el futuro desarrollo de otros sistemas, puede llegar a ser de gran utilidad para explicar los conceptos de seguridad detrás del Navegador. Se sabe que el Browser es un pieza de Software que ha sufrido varios cambios desde la década de los 90, por lo tanto entre los desarrolladores de ésta herramienta ya existen convenciones de qué elementos funcionan mejor. Por consiguiente, no es de extrañar que diferentes browsers estén construidos de formas muy similares, y puedan ser conceptualizados en una Arquitectura de Referencia que manifieste los componentes, mecanismos de comunicación y funciones de esta pieza de Software.

La arquitectura de referencia a construir en este trabajo, fue realizada principalmente a través de la abstracción de las propuestas actuales de los Web Browsers más usados: Google Chrome, Internet Explorer y Firefox (propuesta Electrolysis). Primero se identificarán y analizarán los stakeholders, se identificarán los casos de uso relacionados a cada uno de estos stakeholders y se dará una descripción breve. Luego se presentan 2 patrones que formarán parte de nuestra AR; Browser Infrastructure Pattern donde están los componentes principales, la comunicación con el host y la interacción entre éstos. Y luego el patrón RendererContentTab que muestra el interior del encargado de renderizar una página web. Éstos patrones serán descritos utilizando un template POSA [16] y notación UML para precisar cómo los componentes de la

AR se relacionan.

4.1. Casos de Uso del Browser

4.1.1. Stakeholders (actores) y Concerns de estos

Se es necesario encontrar los actores (definido por sus roles) que participan en el uso y operación del Navegador, estos son:

Browser User (BU)

De este stakeholder depende que se realice el inicio de una petición para buscar una página web, recurso o servicio (Sin éste la utilidad del Web Browser es nula). Una entidad no humana, como un plugin, extensión o instancia de una página web, pueden requerir hacer peticiones por medio de las interfaces del navegador, con la intención de cumplir con los deseo del usuario del host de mostrar el contenido en el Browser.

Host (H)

Crea los procesos necesarios para iniciar el Navegador, además de entregar un ambiente al Browser para que éste pueda funcionar adecuadamente. Ésta identidad se encarga de aplicar políticas de seguridad sobre el Browser cuando se necesiten realizar operaciones o el navegador desee crear nuevos procesos.

Provider(P)

Este puede ser un: Web Server, Web Aplicación, Servicio de actualización del Browser, etc. Su interacción con el Browser se limita a entregar contenido a éste.

4.1.2. Casos de Uso

Los casos de usos y actores están listados a continuación. Notar que lo que se presentarán aquellos casos de uso relacionados exclusivamente con el Browser.

Casos de Uso para Browser User (BU)

El BU representa todas las interacciones posibles del usuario con el Browser y sin él no habría razón para la existencia del navegador; un servidor tampoco existiría dado el mismo principio. Al encontrar los casos de uso de éste stakeholder veremos los concerns de éste, lo que nos permitirá entender las necesidades de seguridad para proteger al navegador. Los casos de uso principales de éste son: Realizar request, Cancelar request y Guardar recurso.

1. Realizar Petición/Request: El caso de uso más importante de éste sistema, se basa en la arquitectura cliente/servidor. Esta acción considera la petición de recursos al host y la búsqueda del recurso bajo la URI dada y la consecuente response del servidor. La descarga no es siempre necesaria, pues existe lo que se conoce como una petición *preflight* donde solo se envía una petición para saber si es seguro realizar una petición/request. Tanto request como response se han condensado en un solo caso de uso. Este caso de uso considera la visualización del recurso en el Host a través del periférico predeterminado.
2. Cancelar request: Un BU pide al Browser detener la búsqueda y todo proceso en transito. Si el recurso es obtenido por el Browser, la detención de éste puede considerar la detención del componente Renderer para mostrar por pantalla la proyección de la página web obtenida.
3. Guardar recurso: Un BU puede necesitar guardar el recurso recientemente adquirido en el sistema de archivos del host (H). El caso de uso inicia una vez que el Browser User haya manifestado el donde desea guardar el recurso.

Casos de Uso para el Host (H)

Los casos de uso del Host representan todas las interacciones que relacionan servicios proveídos al BU al hacer uso de *system calls*, por medio del Browser. El Host debe para esto contar con la respuesta explícita del usuario detrás de BU. Los casos de uso principales de H son: Operación CRUD en el sistema de archivos del host, Monitorear proceso y Pedir recursos.

1. Operación CRUD en el sistema de archivos del host: Acción realizada cuando el Browser obtiene una solicitud explícita o no (automática), del usuario para realizar acciones que involucren permitir el acceso al sistema de archivos para: leer, escribir o modificar/eliminar algún recurso del host.
2. Monitorear proceso: En un browser es normal la comunicación entre los procesos que lo componen y el Sistema operativo por medio de un canal de comunicación, en especial cuando se necesita ver la cantidad de recursos usados.

3. Pedir recursos: Bajo el pedido explícito del usuario, la entidad Browser User, se requieren recursos para iniciar las operaciones deseadas. Estos recursos son obtenidos a partir de llamadas al sistema al host.

Casos de Uso para el Proveedor (P)

La meta del Proveedor es escuchar las peticiones provenientes de diversos tipos de Browser User que necesiten los servicios o recursos del Proveedor. En este escenario tan heterogéneo y distribuido que es la Internet, se enfocará solamente en las peticiones de Browser User. Además, no para toda request del BU habrá un response de P, pues puede pasar que BU esté solamente intentando averiguar si P existe.

1. Escuchar requests: P siempre está escuchando en sus puertos habilitados a posibles peticiones de sus clientes. Una petición es recibida en un formato que P pueda interpretar y en consecuencia realizar las acciones correspondientes que BU le pide.

2. Enviar Response: Por cada solicitud del cliente, en este caso el Web Browser, habrá una respuesta/response por parte del P con la data Resource asociada. Este caso de uso no se realiza, si dentro del header de uno de los paquetes está indicado que la petición fue hecha en modo *preflight*.

3. Crear recurso: Por cada recurso que P crea, una URL única será usada para ser identificada en la Internet. Un Browser User hace uso de tal URL para adquirir el recurso, mientras el host del Proveedor lo deje visible.

4. Configurar opciones: definir distintas medidas en el Proveedor como: acceso a recursos, acceso de Browser users, etc.

Los casos de Uso descritos anteriormente se pueden divisar en la Figura 4.1

4.2. Patrón Browser Infrastructure

4.2.1. Intent

El patron Browser Infrastructure pattern permite la realización de la solicitud o request de un recurso web en la internet por parte de un Browser User, que es un usuario del Host que utiliza un navegador. El patrón permite visualizar la comunicación entre los componentes que conforman el browser así como la comunicación con el proveedor, al que se le realiza el request.

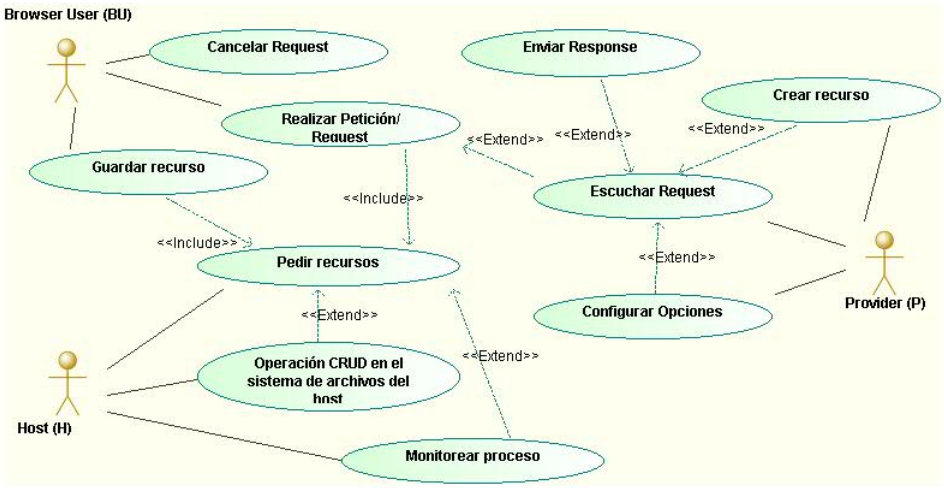


Figura 4.1: Diagrama de Caso de Uso del Web Browser

4.2.2. Ejemplo

Dentro de un host es posible que exista una falta de recursos que el usuario que lo opera necesite. La solicitud de servicios o recursos externos es una de las razones de existir de la Internet. Ésta operación es posible de realizarlo de diferentes maneras, todo depende de lo que el Provider desea entregar.

4.2.3. Contexto

Browser User se encuentra al interior del Host y el Provider es accedido a través de internet. Es común que el contacto se da a través de aplicaciones web, servers que se comunican a través del protocolo HTTP. Para poder visualizar los recursos que el usuario del host puede necesitar y que el proveedor pueda quizás entregar, un navegador web permite completar tal labor de forma transparente al usuario.

4.2.4. Problema

Algunos usuario del Host puede que necesiten obtener recursos desde un Provider, pero el usuario puede que necesite utilizarlos en algún formato en especial o que sean presentados por pantalla para ser mejor visualizados. En este caso, si no se utiliza la herramienta adecuada puede que no sirva de mucho haber conseguido el recurso si no se puede ocupar correctamente. ¿Cómo pueden el Host y Provider estar preparados para esta situación? La solución a este problema debe resolver las siguientes problemáticas:

- Transparencia: el user detrás del Host no se debe preocupar de la maquinaria que existe mientras se realiza una petición a un Provider.
- Estabilidad/Aislación: cada request realizada no debe interferir las con otras.
- Heterogeneidad: Sin importar el tipo de Provider con el que se comunique, el Browser User debe poder comunicarse con cualquier tipo de Provider y también debe poder mostrar adecuadamente al usuario el recurso obtenido.
- Disponibilidad: El user del Host debe ser capaz de poder realizar peticiones en todo momento, sin perder fluidez.

4.2.5. Solución

Un Web Browser puede satisfacer las request del user del Host a través de un Browser User, ya sea por una o varias instancias de Browser User lo que permite tener una diversidad de opciones para navegar por sitios de internet. Un Browser debe ser capaz de poder entregar una navegación rápida y estable, sin que cada sitio accedido afecte a los otros recursos adquiridos.

4.2.6. Estructura

El Browser Client (BC) es la entidad que representa el proceso principal de un Navegador Web y comprende la cantidad mínima de componentes que constituye un Web Browser. Browser Client, Sandbox, GPU Instance (GPUI) y Plugin son del tipo Process (Pr), que residen dentro de un Host (H) con un cierto tipo de sistema Operativo (OS). La mayoría de los navegadores existentes usan un componente céntrico para realizar las operaciones que necesitan afectar al Host del Browser. La figura 4.2 muestra el diagrama del clases para el patrón Browser Infrastructure. Por cada recurso que un BC solicite, se crearán Sandbox (S) que alojarán una instancia del patrón RendererContentTab el que permiten realizar la navegación y posterior visualización del recurso. El componente BC actúa como un broker de las solicitudes de las Sandbox, lo que permite tener un control fino de los mensajes enviados, usando IPC/IPDL/COM, entre los procesos que se comunican. El GPU Instance y Plugins son elementos que pueden comunicarse directamente con el Sandbox, de esta manera cualquier necesidad de recursos del Host pasarán también por el Browser Client. Un usuario que desee hacer un request de un recurso en Internet por medio de un navegador es lo que llamaremos Browser User (BU). Éste usa el Browser Client para realizar peticiones al Provider, donde éste último posiblemente utilice un Web Server para ese cometido (Figura 4.2)

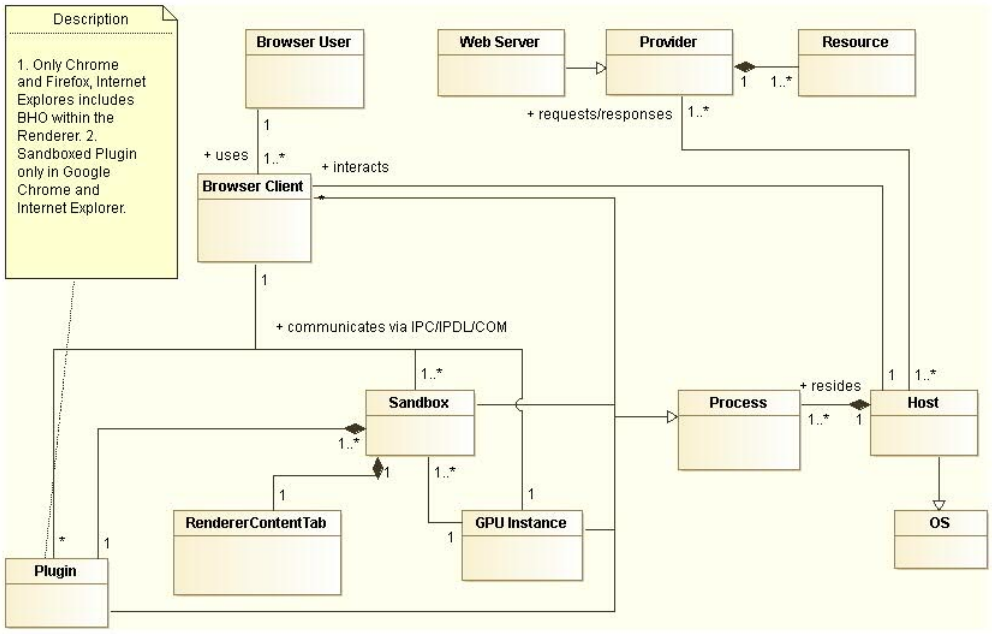


Figura 4.2: Componentes de alto nivel del Browser

4.2.7. Dinámica

Casos de uso relacionados al patrón son los siguientes:

- Realizar request (actor: Browser User)
- Cancelar request (actor: Browser User)
- Guardar recurso (actor: Browser User)
- Recibir Request (actor: Provider)
- Pedir recursos (actor: Host)

Detallaremos el caso de uso Realizar request (Figura X):

Sumario

Un Browser User requiere de abrir un recurso URL que puede ser obtenido mediante el uso del protocolo HTTP, según los requerimientos del Provider. El Browser Client será usado por un Browser User para poder realizar la visualización del recurso URL.

Actor

Browser User

Precondiciones

El Host debe tener uno o más Browser Client para el usuario del Host. Además de estar conectado a una red o Internet. El Provider al que se desea contactar también debe estar disponible.

Descripción

Nota: Los mensajes entre el Browser Client y el Sandbox pueden ser tanto síncronos como asíncronos. No especificaremos en gran detalle , pues lo que importa en este trabajo será el origen y destino de los mensajes (no está dentro del ámbito ver la sincronización).

1. Un Browser User requiere acceder a una URL para obtener cierto recurso de un Provider, para esto usa un Browser Client ya instanciado por el Host. En el interior del Sandbox, existe una instancia del patrón RendererContentTab en su interior.

2. El Sandbox requiere los recursos del Host para obtener lo que hay detrás de la URL. Una petición es realizada desde el Sandbox al Browser Client a través de un canal de comunicación como: IPC, IPDL o COM (dependiendo el tipo de Browser usado), usando la API limitada que posee para comunicarse a un proceso de mayor privilegio.

3. El Browser Client recibe la solicitud, verifica a través de su motor de política-s/normas si la acción del Sandbox puede ser permitida.

4. Si es permita la acción del Sandbox, se utilizará la Network API que contiene el Browser Client para obtener recursos del Host (a través de llamadas al sistema). El Browser Client se comunica internamente con el Host, y este último debe revisar sus políticas y asegurarse que el Browser Client posee el privilegio de hacer la petición del recurso del Host.

5. Si se permite el acceso el recurso, el Browser Client podrá realizar un request a través de la Network API. Si el request no es del tipo *pre-flight*, el Provider recibirá el request y trabajará sobre la petición.

6. El Provider enviará un response al request enviado. Dependiendo como esté implementado el Browser Client, este podría o no tener que esperar a la respuesta (síncrono o asíncrono) del Provider.

7. Una vez obtenida la respuesta esta es almacenada en cache o podría ser guardada dentro del Host.

8. La respuesta del request es enviada por el canal de comunicación al Sandbox del que se originó y posteriormente al RendererContentTab. Si fue recibida una respuesta por parte de la request, el RendererContentTab está listo para preparar el parsing de la página web o utiliza un plugin o GPU para apoyar la visualización del recurso obtenido por la URL. En caso contrario el RendererContentTab dentro del Sandbox creará una página de error.

Curso Alternativo

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

Condiciones póstumas

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

4.2.8. Implementación

- El Sandbox puede ser implementado de diversas maneras. Google Chrome [referencia a página de sandbox de cg] se basa en no reinventar la rueda y utiliza los mecanismos de protección que Windows 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 RendererContentTab. Google Chrome, Firefox e Internet Explorer asumen que los Sandoboxs son procesos que deben regirse 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 BrowserClient, 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 (RendererContentTab en este caso) para dejar aisladas las páginas/recursos de diversos sitios.

4.2.9. Consecuencias

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/Aislación: 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).
- Heterogeneidad: Dado que cada Browser Client trata de seguir los estándares de la W3C, toda página que también siga éstas 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 usuaria pueda mantenerse fluida para el usuario.

Al mismo tiempo el patrón posee las siguientes debilidades:

- Dado que se inician procesos independientes para navegar a un recurso (dependiendo el 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, mostraran su resource de forma incorrecta por el Web Browser.

4.2.10. Ejemplo Resuelto

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 si.

4.2.11. Usos Comúnes

- Google Chrome
- Internet Explorer
- Firefox

4.2.12. Patrones Asociados

- Reference Monitor [Fer13]
- Policy authorization

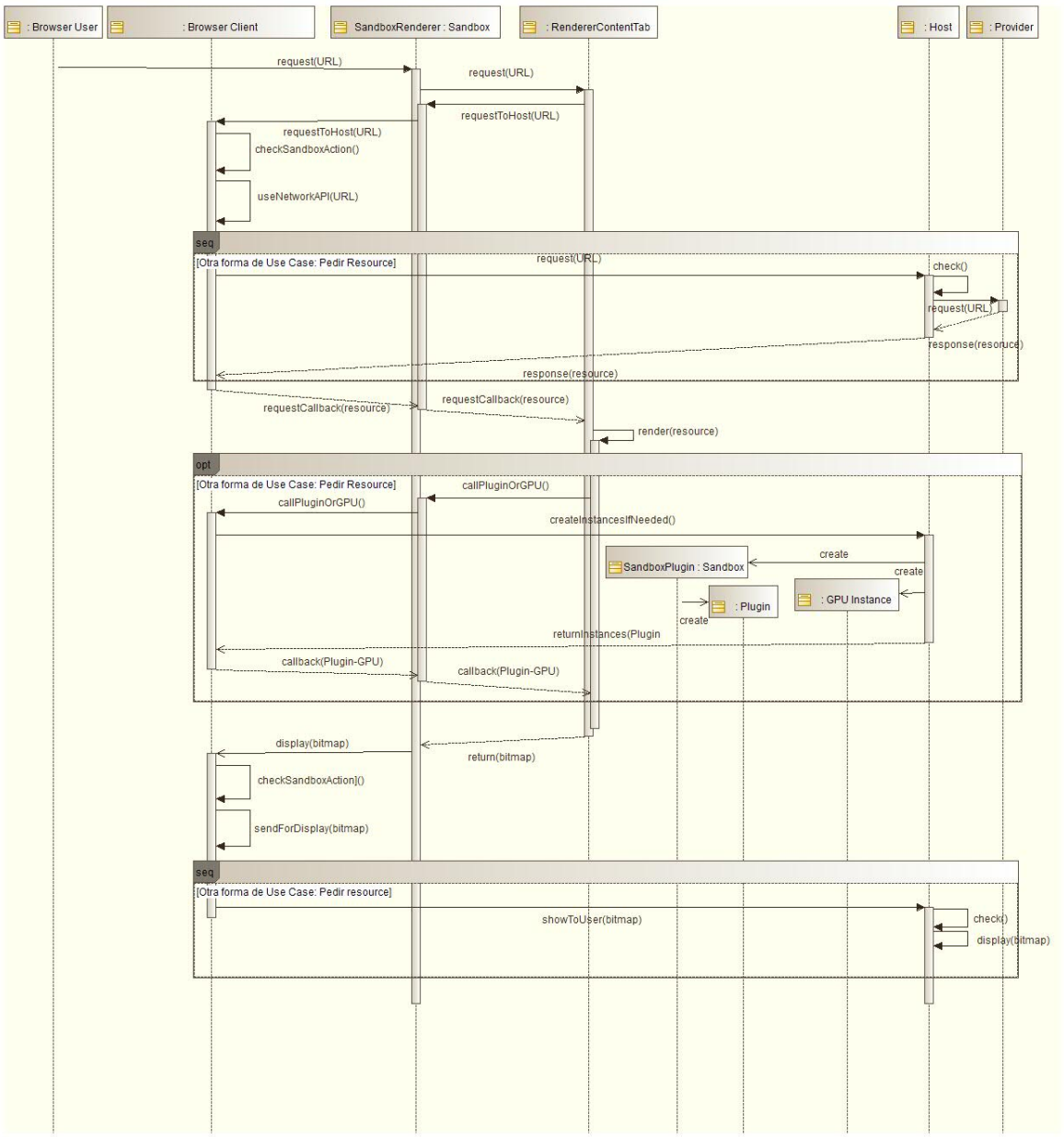


Figura 4.3: Diagrama de Secuencia: Browser User Realiza request

Capítulo 5

Patrones de Mal Uso

5.1. Identificando Amenazas y Patrones de Mal Uso

En este capítulo presentaremos el análisis e identificación de las amenazas dentro del patrón presentado en el capítulo anterior, y 2 patrones de mal uso. Estos artefactos servirán de ejemplo de cómo usar la Arquitectura de Referencia obtenida. El objetivo de éste capítulo es conocer y visualizar los mal usos del sistema, en este caso del Browser, para poder educar a los desarrolladores de proyectos que basan sus sistemas en el uso de éste. Este capítulo realiza los siguientes ítems: Un análisis de amenazas realizado sobre las actividades relacionadas en el caso de uso Realizar request y Recibir Request, luego a partir del resultado del análisis se aplica la metodología explicada en el trabajo de [Bra08] para obtener una lista de amenazas. A través del listado de amenazas es posible detectar o inferir actividades de mal uso que pueden aparecer en uno o más casos de uso, que podrían resultar en una vulneración del sistema. Se presentarán los mal usos de actividades a través de patrones. Se han identificado 2 patrones y serán presentados a continuación.

5.2. Identificando amenazas

La identificación de amenazas es base para poder identificar las medidas adecuadas para poder evitarlas. A través de la metodología de [BRa08] es posible detectar y enumerar las amenazas de manera sistemática al considerar cada actividad en cada caso de uso. Para enumerar las amenazas sistemáticamente los diagramas de actividad ayudan en esta tarea. Usando los casos de uso listado en el capítulo anterior, es posible revelar una lista de amenazas extensas. Solo para sintetizar el proceso, se usará un caso de uso y de éste se extraerá un patrón de mal uso. La figura 5.1

describe el diagrama de actividad que involucra los casos de uso Realizar Request y Recibir Request. Varias amenazas pueden ser reveladas a través de las interacciones visualizadas en el diagrama, por ejemplo: Cuando se realiza la petición al servidor del recurso especificado por la URL, y se entrega otro recurso que no es el pedido.

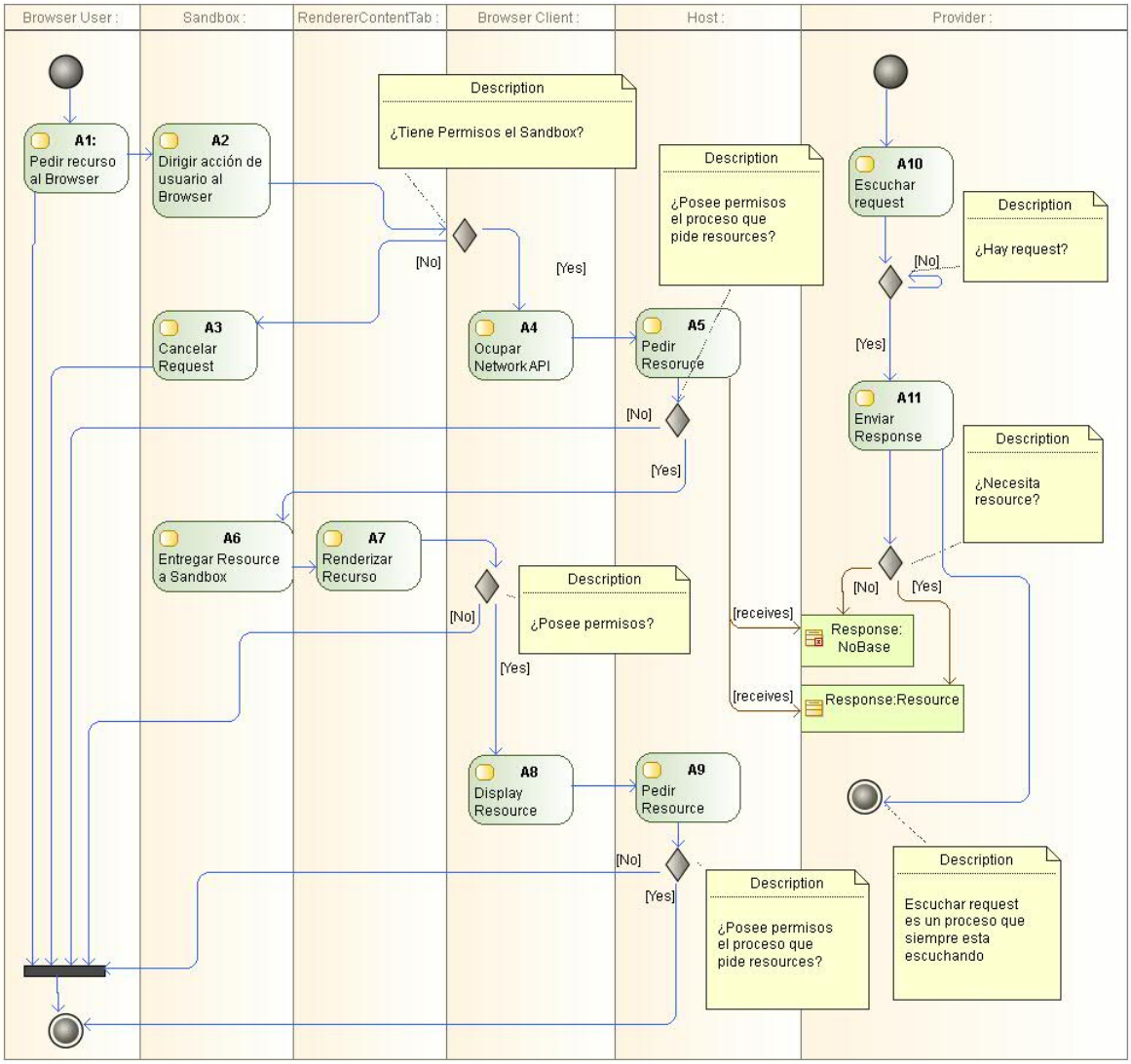


Figura 5.1: Diagrama de Actividad para los casos de uso Realizar Request y Recibir Request

Cuadro 5.1: Tabla con amenazas

Actor	Acción	ID	Sec. Attrib (CO/IN/AV/ACC)	Source (AIn/UIn/Out)	Descripción	Attacker	Asset
Browser User	A1	1.1	CO/IN	Out	Modificación de Tráfico/paquetes	Externo	Browser Client, Host
		1.2	IN/CO	UIn/Out/In	Pedir recurso dentro del Host	Externo	Browser Client, Host
		1.3	CO/IN/AV	UIn/Out	Contactar un Provider Malicioso	Externo	Browser Client, Host
		1.4	CO	Out/UIn	Predecir comportamiento o análisis de tráfico	Externo, Process malicioso	Browser Client
		1.5	CO/IN/ACC	Out	Conseguir contraseñas o información personal	Externo	Browser Client, Host
Host	A5 y A9	5.1	CO	Ain/UIn/Out	Divulgar información	Externo	Browser Client, Host
		5.2	IN/CO	AIn/Out	Imitar a un Host honesto	Administrador Malicioso de Host	Browser Client, Provider
		5.3	CO/IN/AV	Ain/Out/UIn	Contactar Provider Malicioso	Externo	Browser Client, Host
		5.4	AV	Out	Impedir navegación	Externo	Browser Client
Provider	A10	10.1	IN/AV/ACC	Out	Aceptar request de cliente comprometido	Administrador Malicioso, Externo	Provider
		10.2	CO	Ain/Out/UIn	Divulgar información	Externo	Provider

Hemos identificado algunas amenazas al analizar el curso de los eventos de los casos de uso: Pedir Request y Recibir Request (Figura 5.1). La tabla 5.1 hace un sumario del análisis de cada acción en el diagrama de actividad, teniendo en cuenta los atributos de seguridad que podrían ser comprometidos, el origen de la amenaza y los recursos que pueden peligrar. Los atributos de seguridad principales son: Confidencialidad (CO), Disponibilidad (IN), Acontabilidad (AC) e Integridad (IN). El origen de la amenaza puede ser: actor autorizado (Ain), un actor no autorizado (UIn) o un externo (Out). Listar las amenazas que pueden ocurrir en el sistema no es suficiente para entender como los ataques ocurren. Es por esto que utilizaremos 2 patrones de mal uso, que permitirán describir como un ataque ocurre desde el punto de vista del atacante.

5.3. Patrón de Mal Uso: Modificación de tráfico en el Web Browser

En esta sección presentaremos un patrones de mal uso que describe la amenaza encontrada en el patrón Browser Infrastructure que hemos obtenido en este trabajo. Una de las amenazas es que un atacante haya sido capaz de comprometer la respuesta obtenida desde el Provider contactado. El atacante podría tratar de reemplazar algunos parámetros de la respuesta del provider, entregando al Browser User un contenido distinto al original..

5.3.1. Intent

Un atacante podría cambiar contenido o dar uno diferente al esperado cuando el Web Browser (Browser User) recibe la response del Servidor o Provider. Realizando esta acción, el Navegador podría interpretar la información de una manera distinta a que si hubiera recibido el tráfico original.

5.3.2. Contexto

Un Navegador web obtiene resources desde un Provider para poder un Browser User que lo necesita. Un Provider posee resources en forma de páginas web, servicios web u otro tipo. El Provider generalmente consiste en una Web App o Web server, que puede permitir entrada y salida de datos a otras aplicaciones, normalmente éstos están construidos usando HTML, Javascript y CCS. Sea que éste desea entregar servicios, como también otros deseen conectarse a él, todo tipo de comunicación estará encima del protocolo HTTP. Un Provider, dependiendo del tipo, puede llegar a recibir muchas peticiones de diversos Host para obtener resources de éste. Dependiendo del tipo de

peticiones, estas pueden o no ser permitidas. Aquellas que son permitidas, el Provider generará una response a la request del Host, la cuál terminará llegando de vuelta (o no) al Browser Client que generó el request.

5.3.3. Problema

¿Cómo podría un atacante engañar al Browser User y Host, por medio de la modificación de tráfico entre las entidades participantes en la comunicación? Es posible que un atacante esté escondido entre medio de la comunicación que hay entre el Host y el Povider, lo que resulta en que el contenido modificado podría afectar dentro del RendererContentTab o el Browser Client. Esto en consecuencia podría traer diferentes resultados, desde el robo de información privada del Browser User que utiliza el Browser Client para personalizar la navegación como también los token de autenticación hacia los Providers que el Browser User ha utilizado previamente. Dependiendo del tipo de atacante, es posible que este pueda incluso afectar al Host del Browser, dejando que el atacante pueda realizar actos maliciosos con los recursos del Host. El ataque puede sacar ventaja de las siguientes vulnerabilidades:

- El origen u **origin** que define el Same Origin Policy que hace cumplir el Browser difiere en cada tipo de Web Browser [50, 51, 52, 53, 54].
- El **origin** no es suficiente como método de aislación entre resources (páginas web, scripts, css y otros) [55, 56, 57, 58].
- Cualquiera puede crear una extensión o plugin, etc. para algún tipo de Browser y hacerlo pasar como algo inofensivo, pero el usuario no se daría cuenta que un Man in the Browser podría estar ocurriendo [59, 60, 58, 61]. Éste programa, posiblemente malicioso podría, afectar el tráfico sin que queden rastros, pues tiene permisos del usuario al ser éste mismo quién autorizó su instalación.
- Es posible afectar al Browser Client y en consecuencia al Host, sin tener que buscar vulnerabilidades en el Web Browser. Solamente utilizando métodos de ingeniería social, sobre el usuario del Host basta para lograr un ataque, pues es "the weakest link".
- La arquitectura para extender la funcionalidad del navegador, a través de extensiones, plugins y otros, depende demasiado del manufacturador. Y probablemente posee una gran superficie de ataque.

El ataque puede ser facilitado por:

- Existen muchas herramientas para realizar ataques de ingeniería social, lo que permite hacer que el Browser User acepte realizar la instalación de extensiones o plugin maliciosos más fácilmente.

- Cualquier script basado en texto puede ser usado para explotar el intérprete del navegador web. Muchas veces también es posible utilizar los mismos elementos del lenguaje de scripting para poder pasar ciertas barreras de seguridad entregadas por el SOP, pues el lenguaje es basado en prototipos.
- Los manufacturadores de Navegadores no poseen aún mecanismos de defensa que permitan identificar efectivamente cuando un resource puede ser malicioso.
- Los métodos de encriptación no pueden hacer nada contra un ataque que puede modificar el tráfico antes de enviar el mensaje o después de recibir el mensaje.
- No existen mecanismos de defensa estandarizados, pues cada manufacturador realiza lo necesario para la marca a la que se acopla. Si el Host posee muchos browser, la superficie de ataque es bastante mayor.

5.3.4. Solución

La solución estructural usada es la misma creada por el patrón Browser Infrastructure que se revisó en el capítulo 4. La clase Attacker es cualquier entidad que podría llevar a cabo una acción arriesgada contra la integridad del Browser, usuario, Host y Provider (Figura 5.2). El atacante es capaz tanto de interceptar las response que el Provider envía al Browser Client usando al Host como receptor, o haciendo que el Browser realice modificaciones en el input que va hacia el Provider, utilizando scripts u otro resources.

5.3.5. Dinámica

En la figura 5.3 se muestra la serie de pasos necesarios, para realizar uno de los tantos mal usos que se pueden realizar durante el caso de uso Realizar Request. El atacante queda entre el Browser Client y el Host, interceptando la realización del request original y modificando el tráfico a su gusto; usualmente un ataque basado en este mal uso se le llama Man-in-the-Browser (MITB) [58, 61, 60, 59]. Esto podría perfectamente suceder cuando el Browser User ha permitido la instalación de plugins, extensiones o programas externos en el Host y Browser Client [ref]

Sumario

El atacante intercepta el tráfico entre Host y Browser Client.

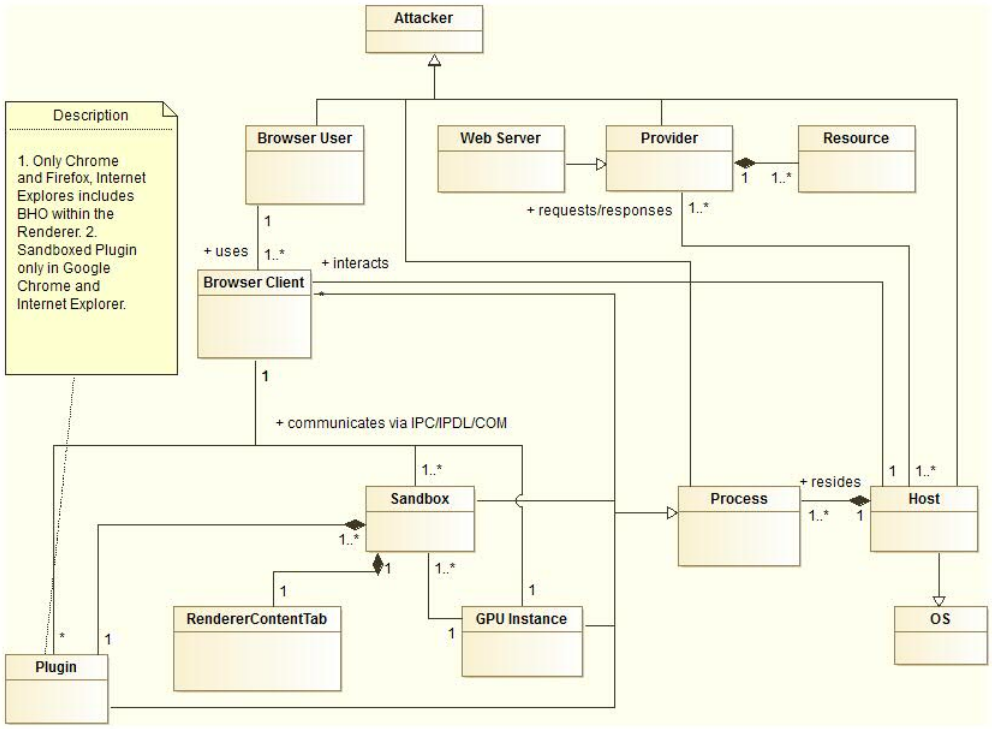


Figura 5.2: Diagrama de Clases para el patrón de Misuse

Actor

Atacante

Precondiciones

Para que el ataque pueda pasar desapercibido, es necesario que el Browser User o el usuario detrás del browser haya caído primero en un ataque de ingeniería social o el atacante haya podido instalar directamente un process malicioso en el Host directamente.

Descripción

1. Un atacante utiliza alguna técnica de ingeniería social o vulnerabilidad en el sistema, para crear una entidad que se encargará de estar entre medio del Browser Client y el Provider. Para esto realiza el caso de uso Pedir Resources para que el proceso, plugin o extensión se aloje en el Host.

2. Un Browser Consumer desea hacer un request a cierta URL, por lo que los

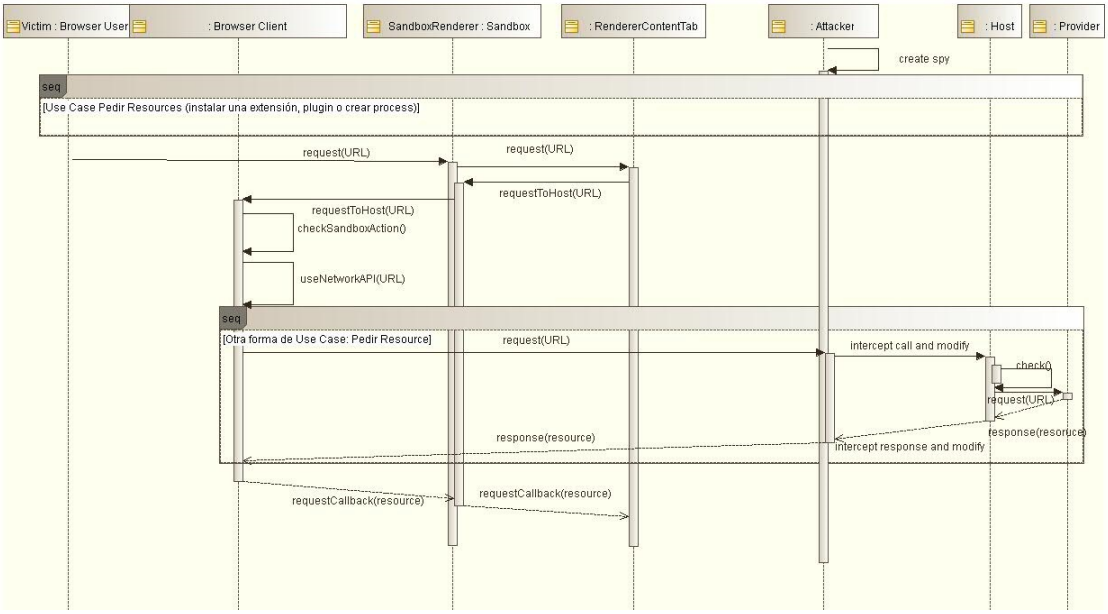


Figura 5.3: Diagrama de Secuencia para el Mal uso: Modificación de tráfico en el Web Browser

primeros pasos de Realizar Request son similares.

3. En el momento en que el Browser Client va a realizar la llamada al sistema para enviar el mensaje del Host al Provider, el Browser Client llamará al plugin, extensión o process malicioso, pues el Browser Client ha sido interferido para que realice esa acción.

4. El atacante entonces recibirá todo el tráfico del Browser Client, con el cuál podría modificar a su antojo.

5. Finalmente la víctima está totalmente comprometida.

Post condiciones

La victima quedará completamente comprometida y probablemente no sea posible detectar la alteración del mensaje, pues también es posible que se modifiquen los log del Host.

Usos Conocidos

El Web Browser es un software que posee diversas implementaciones, por lo tanto la cantidad de vectores de ataque es significativa. Algunos de éstos son:

- Una extensión basada en la arquitectura de Chrome o la API WebExtension Firefox, podría interceptar los datos antes que llegue al Browser Client [54] o dado a una vulnerabilidad del mismo elemento un atacante se está aprovechando de su funcionalidad para realizar ataques [58, 61]. Dado que el Plugin, la extensión o el process son elementos que el Host confía, es posible que pase indetectable el ataque y además los métodos de encriptación no sirven como medida de mitigación.
- Éste tipo de ataque puede ser usado como base para otros ataques más avanzados. Un ejemplo es cuando el Browser posee vulnerabilidades *cross-origin javascript capability leaks*, donde los diferentes modelos de seguridad usados por Javascript y el DOM pueden interferir entre si, causando que una petición *cross-origin* se pueda realizar aún cuando SOP debería ser capaz de detener tal ataque [56].
-

5.3.6. Consecuencias

El mal uso tiene las siguientes consecuencias para el Attacker:

- Objetivos: pueden ser diversos, destacándose el vandalismo, personificar a otra persona u obtener una ganancia monetaria. Mientras el atacante se pueda interponer entre el Host y el tráfico que se envía al Provider, la confidencialidad e integridad de los datos está completamente perdida. La privacidad del usuario ya no se puede asegurar tampoco.
- Silencioso: Dado que el atacante ha logrado interponerse entre las llamadas de sistema que se realizan al Host para enviar los datos al Provider, el Host no reconocerá ni logueará ninguna anomalía. Las llamadas hechas al Host son totalmente legales y nada fuera de lo normal para éste.
- El atacante podría realizar acciones que afecten la integridad del Host.

Posibles fuentes de fallo:

- Si el Browser User es capaz de evitar o ignorar el ataque de Ingeniería Social realizado al comienzo, no existiría este mal uso. Esto debe considerar que el usuario también no se encuentre con páginas o contenido malicioso, que podrían afectar otro componente de Browser, pero que causarían en el mismo efecto del Mal Uso señalado.

5.3.7. Contramedidas

Para prevenir este tipo de mal uso se recomienda tomar las siguientes medidas preventivas:

- Servicios de Reputación como Smart Screen de Internet Explorer y Download Application de Google Chrome, ayudan a identificar páginas y contenido/resources que podrían contener malware que se instale como plugins, entensiones o process en el Host del Browser User.
- Entregando educación sobre los peligros de navegar por internet y aclarar al usuario que él es la última línea de defensa contra éste tipo de ataques.
- White y Black list instaladas en los browser son una medida preventiva para evitar la navegación en páginas o recursos maliciosos ya conocidos.
- Navegadores como Google Chrome e Internet Explorer ofrecen el Sandboxing. Éste mecanismo de defensa limita las acciones del atacante, que pudieran afectar la integridad del sistema.

5.3.8. Evidencia Forense

¿Dónde es posible encontrar evidencia? Dependiendo de lo deseado por el atacante, las acciones que cometerá pueden diferir. Sin embargo el log interno del browser debería de poder servir para auditar el sistema, esto gracias a que mientras no se haya encontrado una vulnerabilidad en el Sandbox del Browser, el atacante no puede borrar completamente sus huellas.

5.3.9. Patrones relacionados

- En el patrón Browser Infrastructure, el Browser Client actúa como el Reference Monitor explicado en [62].

Bibliografía

[1] Global Stats StatCounter. Top 5 desktop, tablet and console browsers, 2015.

[2] World Wide Web Consortium W3C. About.

[3] Karen M Goertzel, Theodore Winograd, Holly L McKinley, Lyndon J Oh, Michael Colon, Thomas McGibbon, Elaine Fedchak, and Robert Vienneau. Software security assurance: A state-of-art report (sar). Technical report, DTIC Document, 2007.

[4] J. Yoder, J. Yoder, J. Barcalow, and J. Barcalow. Architectural patterns for enabling application security. *Proceedings of PLoP 1997*, 51:31, 1998.

[5] Eduardo B Fernandez. A methodology for secure software design. In *Software Engineering Research and Practice*, pages 130–136, 2004.

[6] Bill Whyte and John Harrison. State of Practice in Secure Software: Experts' Views on Best Ways Ahead. IGI Global.

[7] Carnegie Mellon University Computer Emergency Response Team. Early identification reduces total cost (segment from cert's podcasts for bussiness leaders).

[8] Mike Hicks. Interview to **Kevin Haley** (from **Symantec**), 2014. Mike Hicks (Profesor of Software Security course in Coursera.org).

[9] Eduardo Fernandez-Buglioni. *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons, 2013.

[10] M.M. Larrondo-Petrie, K.R. Nair, and G.K. Raghavan. A domain analysis of web browser architectures, languages and features. In *Southcon/96. Conference Record*, pages 168–174, Jun 1996.

[11] Alan Grosskurth and Michael W. Godfrey. A reference architecture for web browsers. pages 661–664, 2005. URL: `http://grosskurth.ca/papers.html#browser-refarch`.

[12] Alan Grosskurth and Michael W. Godfrey. Architecture and evolution of the modern web browser. URL: `http://grosskurth.ca/papers.html#browser-archevol`. Note: submitted for publication.

[13] Eduardo B Fernandez, Michael VanHilst, Maria M Larrondo Petrie, and Shihong Huang. Defining security requirements through misuse actions. In *Advanced Software Engineering: Expanding the Frontiers of Software Technology*, pages 123–137. Springer US, 2006.

[14] Fabricio A Braz, Eduardo B Fernandez, and Michael VanHilst. Eliciting security requirements through misuse activities. In *Database and Expert Systems Application, 2008. DEXA'08. 19th International Workshop on*, pages 328–333. IEEE, 2008.

[15] Paris Avgeriou. Describing, Instantiating and Evaluating a Reference Architecture: A Case Study. *Enterprise Architect Journal*, 342(1):347, 2003.

[16] Frank Buschman, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. A system of patterns: pattern-oriented software architecture, 1996.

[17] IEEE Cyber Security. Avoiding the top 10 security flaws.

[18] The MITRE Coporation. Common vulnerabilities and exposures.

[19] Jeremiah Talamantes. The social engineer's playbook.

[20] Wade Alcorn, Christian Frichot, and Michele Orrù. *The Browser Hacker's Handbook*. John Wiley & Sons, 2014.

[21] 2014 Draft XMLHttpRequest Level 1. Xmlhttprequest specification.

[22] The WebSocket API. Websocket specification.

[23] WebRTC 1.0: Real time Communication Between Browsers. Webrtc specification.

[24] W3C Working Group. Html5 specification.

[25] Adam Barth, Collin Jackson, and William Li. Attacks on javascript mashup communication. In *Proceedings of the Web*, volume 2. Citeseer, 2009.

[26] Michal Zalewsk. Browser security handbook, part 2. Web page, Google, 2008.

[27] Adam Barth, Collin Jackson, and John C Mitchell. Securing frame communication in browsers. *Communications of the ACM*, 52(6):83–91, 2009.

[28] Bryan Sullivan and Vincent Liu. *Web application security*. McGraw-Hill, 2012.

[29] Adam Barth, Collin Jackson, Charles Reis, TGC Team, et al. The security architecture of the chromium browser, 2008.

[30] Open Security Architecture. Definitions by osa.

[31] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley Professional, 3rd edition, 2012.

[32] A framework for analysis and design of software reference architectures. *Information and Software Technology*, 54(4):417–431, April 2012.

[33] Oscar Encina. Towards a Security Reference Architecture for Federated Inter-Cloud Systems. 2014.

[34] Matthias Galster and Paris Avgeriou. Empirically-grounded Reference Architectures: A Proposal. pages 153–157, 2011.

[35] Keiko Hashizume, Eduardo B Fernandez, and Maria M Larrondo-petrie. A reference architecturefor cloud computing. Submitted for publication. 2014.

[36] Eduardo Fernandez, Juan Pelaez, and Maria Larrondo-Petrie. Attack patterns: A new forensic and design tool. In *Advances in digital forensics III*, pages 345–357. Springer New York, 2007.

[37] E.B. Fernandez, N. Yoshioka, and H. Washizaki. Modeling misuse patterns. In *Availability, Reliability and Security, 2009. ARES '09. International Conference on*, pages 566–571, March 2009.

[38] N Yoshioka. A development method based on security patterns. *Presentation, NII, Tokyo*, 2006.

[39] Nobukazu Yoshioka. Integration of attack patterns and protective patterns. In *1st International Workshop on Software Patterns and Quality (SPAQu'07)*, page 45, 2007.

[40] Juan C Pelaez, Eduardo B Fernandez, and Maria M Larrondo-Petrie. Misuse patterns in voip. *Security and Communication Networks*, 2(6):635–653, 2009.

[41] Eduardo B Fernandez, Nobukazu Yoshioka, and Hironori Washizaki. A worm misuse pattern. In *Proceedings of the 1st Asian Conference on Pattern Languages of Programs*, page 2. ACM, 2010.

[42] Keiko Hashizume, Nobukazu Yoshioka, and Eduardo B Fernandez. Misuse patterns for cloud computing. In *Proceedings of the 2nd Asian Conference on Pattern Languages of Programs*, page 12. ACM, 2011.

[43] Jaime Muñoz-Arteaga, Eduardo B Fernandez, and Héctor Caudel-García. Misuse pattern: spoofing web services. In *Proceedings of the 2nd Asian Conference on Pattern Languages of Programs*, page 11. ACM, 2011.

[44] Eduardo B Fernandez, Ernest Alder, Richard Bagley, and Swati Paghdar. A misuse pattern for retrieving data from a database using sql injection. In *BioMedical Computing (BioMedCom), 2012 ASE/IEEE International Conference on*, pages 127–131. IEEE, 2012.

[45] Ali Alkazami and Eduardo B Fernandez. Cipher suite rollback: A misuse pattern for the ssl/tls client/server authentication handshake protocol. 2014.

[46] Oscar Encina, Eduardo B Fernandez, and Raúl Monge. A misuse pattern for denial-of-service in federated inter-clouds. 2014.

[47] Ma Rajab, Lucas Ballard, and N Lutz. CAMP: Content-agnostic malware protection. *Proceedings of Annual ...*, 2013.

[48] Top 10 2013 - OWASP.

[49] Security/ProcessIsolation/ThreatModel.

[50] W3C. Same Origin Policy. Web page, W3C, 2010.

[51] Charles Reis and Steven D Gribble. Isolating web programs in modern browser architectures. *Proceedings of the fourth ACM european conference on Computer systems EuroSys 09*, 25(1):219, 2009.

[52] Collin Jackson and Adam Barth. Beware of finer-grained origins. *Web 2.0 Security and Privacy*, 2008.

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

[54] Stefano Di Paola and Giorgio Fedon. Subverting Ajax. *23rd Chaos Communication Congress*, (December), 2006.

[55] Marin Silic, Jakov Krolo, and Goran Delac. Security vulnerabilities in modern web browser architecture. *MIPRO, 2010 Proceedings of the 33rd International Convention*, 2010.

[56] Adam Barth, Joel Weinberger, and Dawn Song. Cross-Origin JavaScript Capability Leaks : Detection , Exploitation , and Defense. *Opera*, 147:187–198, 2009.

[57] Mark Vincent Yason. D Iving I Nto Ie 10 ' S E Nhanced P Rotected M Ode S Andbox.

[58] L Liu, X Zhang, G Yan, and Songqing Chen. Chrome extensions: Threat analysis and countermeasures. *... of the Network and Distributed Systems ...*, 2012.

[59] Timothy Dougan and Kevin Curran. Man in the Browser Attacks. *International Journal of Ambient Computing and Intelligence*, 4(1):29–39, 2012.

[60] 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*, pages 110–119, 2009.

[61] Adam Barth, Adrienne Porter Felt, Prateek Saxena, and Aaron Boodman. Protecting Browsers from Extension Vulnerabilities. *Ndss*, 147:1315–1329, 2010.

[62] Eduardo B Fernandez and Rouyi Pan. A pattern language for security models. *proceedings of PLOP*, 1:1–13, 2001.