A Reference Architecture for Web Browsers: Part III, A pattern for a Web Browser Kernel.

Paulina Silva and Raúl Monge, Universidad Técnica Federico Santa María
Eduardo B. Fernandez, Florida Atlantic University

Web Browsers are a fundamental component of the Internet and have critical value for security. Since 2000 a number of new designs have appeared, because the **Monolithic Architecture** used before had not considered some important design decisions, like security, stability and others. Modern Web Browser architecture designs are likely to be based on operating systems properties, where several cooperative processes use the same process structure used by operating systems to implement system services. A **Web Browser Kernel or WBK** is the main component representation in charge of controlling what the Web Browser does. It communicates with other components, now most likely child processes, and sends instruction to them; of course, following a **Browser User**'s intentions. A **Web Browser Kernel or WBK** describes the module in charge of the **main control flow** in a Web Browser and we present here a pattern for this kind of system.

Categories and Subject Descriptors: [**Information systems**]: Browsers—; [**Software and its engineering**]: Patterns—

Additional Key Words and Phrases: Browser, Web Client, Browser Renderer, Reference Architecture, Pattern

ACM Reference Format:

Paulina Silva, Raul Monge, and Eduardo B. Fernandez. A Reference Architecture for Web Browsers: Part III, A pattern for a Web Browser Kernel HILLSIDE Proc. of Conf. on Pattern Lang. of Prog. 24 (October 2017), 10 pages.

1. INTRODUCTION

A Web Browser is mainly defined by its ability to obtain resources (images, videos, texts, etc.) from the Internet. The main components built into the Web Browser are: the **Browser Engine or BE** and the **Rendering Engine or RE**. The first controls the **main action flow** of the Web Browser such as: content retrieval, management of bookmarks, reception of user's input, system calls made to the operating system to write/read into the filesystem, etc., meanwhile the second component retrieves resources and parses them to show to the user (**rendering flow**). Tradicionally, Web Browsers had a **Monolithic Architecture** that combined a **RE** and a **BE** into a single process image. Old version of Internet Explorer, Firefox and Safari, executed the Web Browser in a single operating system protection domain. By design this architecture was not only insecure, because it could run code with the **Browser User**'s privileges and write/read the filesystem, but it was also difficult to maintain and slow to obtain resources as technologies or tools like AJAX, Web Workers or tabs emerged. Modern Web Browser on the contrary, use a **Modular Architecture** where they separate the **BE** and **RE**; leaving the **RE** with lower privileges than the **Browser User**'s. There are different types of designs documented in [Barth et al. 2008], but Chrome and the OP Browser [Grier et al. 2008] which are similar, run with multiple instances of **RE**s, each in a separate protection domain. The design of a Web Browser in a **Modular Architecture** is mainly driven by the **RE**, since it is responsible for most parsing and decoding tasks; Historically in a **Monolithic Architecture**, these tasks have been the source of a large number of Web Browser vulnerabilities and crashes [Barth et al. 2008]. In this document we introduce an architectural pattern that describes **the heart or main module** of Web Browsers also called **Browser Engine or BE**, the **Web Browser Kernel or WBK**.

1.1 Patterns and Related Work

We have used patterns in our work because they encapsulate solutions to recurrent problems and define a way to concisely express requirements and solutions, as well as providing a communication vocabulary for designers [Gamma et al. 1994; Buschman et al. 1996]. The description of architectures using patterns makes them easier to understand, provides guidelines for design and analysis, and can define a way of making their structure more secure. We have looked at several Web Browsers, like Google Chrome, Firefox and Internet Explorer (Edge) to abstract the main components and interactions, and write them as a pattern. We do not say that ours is the optimal or the only solution for this problem. However, this solution is representative of several traditional and modern Web Browsers and it is a generic solution that can be used again, in other words, it is a pattern in itself.

We are currently developing a series of architectural patterns to build a Reference Architecture (RA) and a Security Reference Architecture (SRA) for a Web Browser [Silva et al. 2016b; Silva et al. 2016a; Silva et al. 2016c]. An RA is an abstract architecture that describes functionality without getting into implementation details. Its aim is to provide a guide to build architectures for concrete versions of some system or to extend such system [Avgeriou 2003; Galster and Avgeriou 2011; Angelov et al. 2012]. A SRA, provides information about possible threats and defense mechanisms of the system. [Fernandez et al. 2016], proposed an approach for building SRAs using patterns, where they defined a precise and semiformal security cloud computing architecture for the complete cloud environment. This work showed that SRAs are useful to apply security to cloud systems and for a variety of other purposes. Our intent in describing the Web Browser as an RA and SRA is to understand the symbiosis between the client and the Web Server in which the **Browser User** asks for resources, and its related security implications. Because we know that a client would not exist without its counterpart, the server, we think that security must be seen and understood from both sides and not only on the server side as usually done [Alcorn et al. 2014].

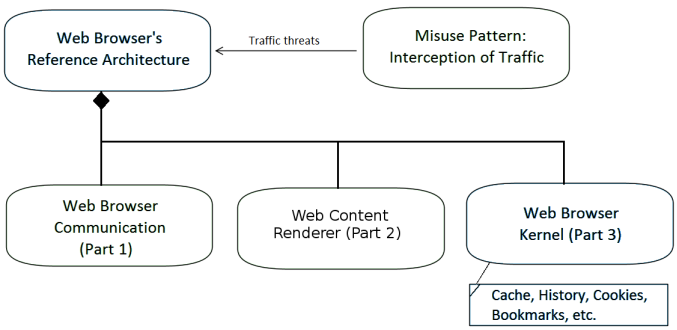

Fig. 1: Pattern Diagram of our current work on the Reference Architecture for the Web Browser.

To understand the construction of our RA, we describe it as a pattern diagram (Figure 1). This pattern diagram shows the relationships between our previous patterns [Silva et al. 2016b; Silva et al. 2016a; Silva et al. 2016c], rounded rectangles represent patterns and the arcs indicate dependencies between patterns. Figure 2 [Silva et al. 2016b] shows an overview of the high level components of a **modern Web Browser** and how they are related to the current pattern, **Web Browser Kernel or WBK**. A **Browser User** using a Web Browser requests pages to a **WBK** that subsequently sends HTTP/S requests to one or more **Providers**/Web Servers for the needed resources. Meanwhile resources are coming to the Web Browser, the **WBK** delegates the parsing and rendering process to its children processes, which are **Controlled Process** instances. To send each requests to the correct **Controlled Process**, a **Reference Monitor or RM** should verify the request's **Origin**, so later they do not intervine each other.

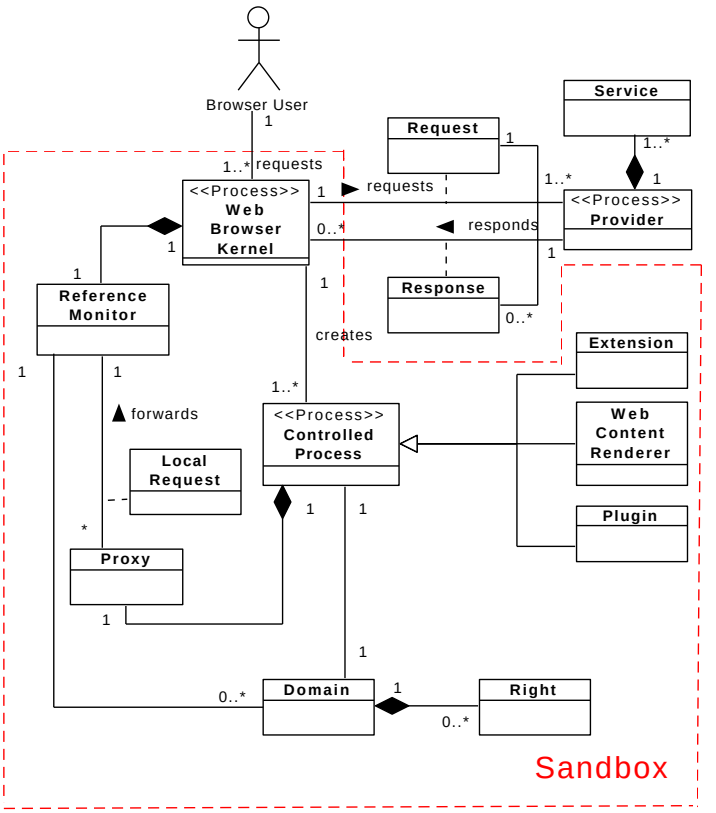

Fig. 2: Overview of components for a modern Web Browser (with a Modular Architecture)

Our pattern is intended for Web Browser architects, security analysts and web developers that are interested on implementing traditional or modern Web Browsers with either **Monolithic Architecture or Modular Architecture**.

2. WEB BROWSER KERNEL OR WBK

2.1 Intent

A **WBK** provides a main component or main control point to: secure and orchestrate/control/supervise actions and communication channels among the other components of the Web Browser to satisfy the needs of a **Browser User** surfing the web (Figure 2).

2.2 Context

A Web Browser is a client which searches for resources on Web Servers connected to the Internet. To display requested resources, the Web Browser components must have enough allocated system resources; an interface must be displayed and the component subsystems of a Web Browser need to be instantiated and assembled together so they are ready to respond to user inputs and provide the corresponding functionality. Faults in components can also occur and should be handled. A central piece is needed to obtain the **Browser User**'s privileges to work and keep the **main action flow** of the Web Browser in control.

2.3 Problem

Mainly two types of architectures could be used to implement a Web Browser: **Monolithic Architecture** or **Modular Architecture**. If the Web Browser has a **Monolithic Architecture**, it may not be able to satisfy all the requests and sometimes these requests could fail, making the Web Browser crash [Wu 2014; Barth et al. 2008]. A **Modular Architecture** on the contrary, allows communication and control features that the **Monolithic Architecture** does not. Using a **Modular Architecture** approach, security, interoperability, maintainability, reliability and performance concerns could be addressed in nowadays modern Web Browsers. Regardless of the architecture, a Web Browser must have a central piece of software that must organize and orchestrate/control/supervise a set of functional subsystems within it that collectively implement browsing features to surf the Internet. How can we control the flow of actions of a Web Browser?

The solution to this problem must resolve the following forces:

—**Separation of concerns:** Since the Web Browser has the **Browser User** privileges, only the most important and sensitive actions should be allowed to access to the computer system resources.

—**Security:** For different types of Web Browser architectures, security should be easy to introduce into the Web Browser.

—**Fined-Grained Control:** Depending on the actions needed, communication between the Web Browser's components should be controlled.

—**Performance:** A resource should be displayed to the **Browser User** as fast as possible and respond to **Browser User**'s interaction as quickly as possible to ensure a better browsing experience.

—**Reuse/Modularization:** It should be possible to implement it in different types of Web Browser architectures, since every type of Web Browser needs a central module to start the **main action flow**.

2.4 Solution

The solution to this problem is a structure already implemented on traditional and modern Web Browsers: the **Browser Engine or BE** [Vrbanec 2013; Wu 2014; Barth et al. 2008]. In this work we will call the BE: **Web Browser Kernel or WBK**, and is in charge of the **main action flow** of the Web Browser and delegates the rendering work, **rendering flow**, to the **Rendering Engine/s**. In our pattern solution Figure 2, the WBK acts as the parent component or central control point of the Web Browser and will work with **Controlled Processes**, mainly **Web Content Renderer or WCR** [Silva et al. 2016b] instances, to display accordingly the content being asked.

2.4.1 *Structure.* Figure 3 shows the class diagram for the **Web Browser Kernel** pattern. The Web Browser's host machine receives the **Browser User** interactions from input devices like mouse or keyboard. When an url is prompted from the **Browser User**, the **windows manager** of the host machine delegates this action to the integration point **UI or User Interface** of the **WBK**. Then, the **Browser Window** searches in the **Cache Storage** for an old version of the page. If no old copy of the page from the requested url is found, an implementation of the **Network Stack** is used to make system calls for the requested url. **User Storage** is used to complete the **Browser User**'s information in the outgoing request, like adding cookies, etc. **Browser Storage** is in charge of storing binaries or files downloaded by the Web Browser. A **Security library** component is implemented on the Web Browser in order to send requests in a secure fashion; maintaining privacy, confidentiality, and integrity of the **Browser User**'s requests. The **Password Manager** is in charge of securing old passwords from the **Browser User** and use them in case they are needed. **History Manager** saves all past requests from the **Browser User**, and lets its **Browser User**s modify them at will. When the request is finally encapsulated to make the request to a server, sockets from the operating system will be used to send the request to the Web Server of interest. Once a response to the request comes back to the host machine, the **Browser Kernel** will receive it and the **Browser**

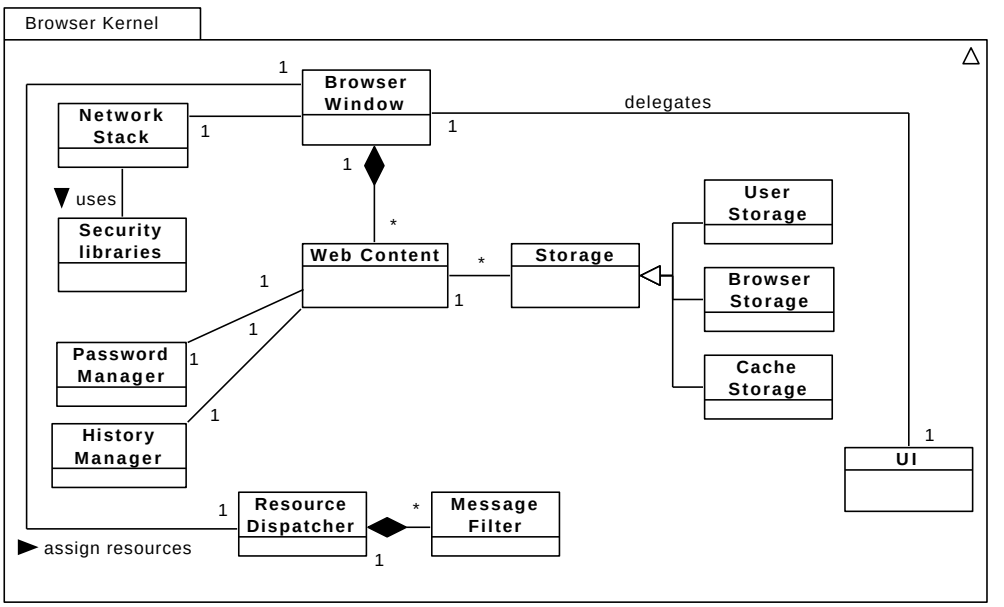

Fig. 3: Class diagram of the Web Browser Kernel (WBK)

Window will use the **Resource Dispatcher** to send it it to a new or old **Web Content Renderer**, through the **Message Filter**.

2.4.2 *Dynamics.* Some use cases for the **main action flow** are the following:

—Content Retrieval

—Save to storage: cache, Web Browser data or user data

—Manage bookmarks

—Monitor Web Browser

—Receive user input

—Send content to be rendered

Below we show in detail the Content Retrieval use case, since it is the most important use case for the Web Browser.

2.4.3 *Summary.* The **Browser User** asks for the content indicated by the URL typed on the keyboard, or interacts with an already loaded resource on the Web Browser.

2.4.4 *Actor.* **Browser User** (Figure 2)

2.4.5 *Preconditions.* The host machine of the Web Browser must be connected to the Internet.

2.4.6 *Description.* When a **Browser User** surfs the Internet, the **WBK** receives the necessary input, like **Browser User** interaction through the keyboard or the mouse. Figure 4 shows the corresponding scenario.

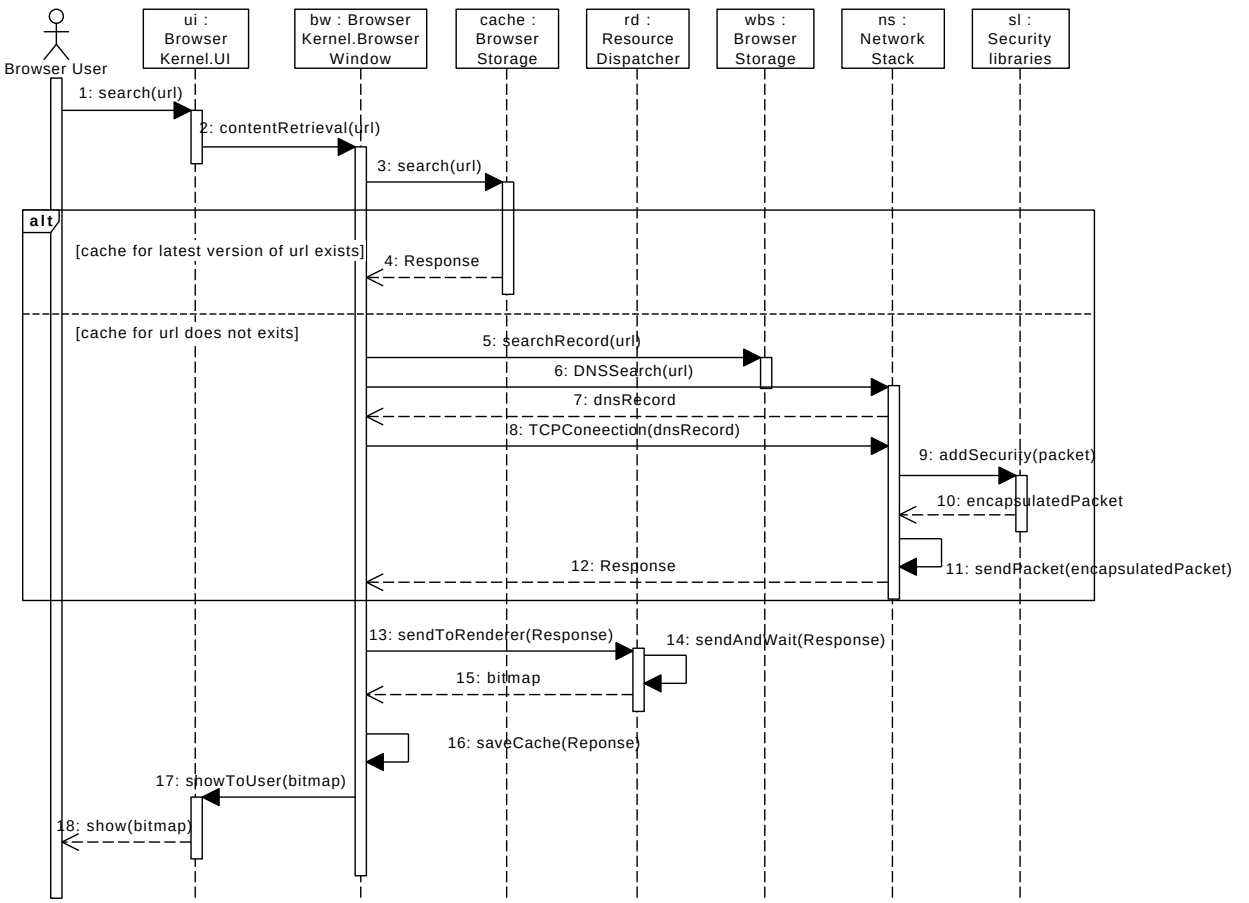

Fig. 4: Dynamics for: Content Retrieval

—(Step 1) A **Browser User** that wishes to retrieve content, interacts with the Web Browser and this interaction is received by the interface provided by the **WBK's User Interface (UI)**, which receives in this case the url of the page the **Browser User** wishes to see.

—(Step 2) The **Browser Window** receives input from **Browser User**s with the help of its **UI**.

—(Steps 3-4) Before searching the content on the Internet, the Web Browser will search in its cache if recent data on the requested url has been saved. If the content is available and up to date, the Web Browser will render it. Go to step 13.

—If the Web Browser did not find anything in the cache for the asked url.

—(Steps 5-7) The **Browser Window** will ask the **Browser Storage** for the **Browser User**'s credentials if needed and will ask the **Network Stack** for a DNS search of the url.

—(Steps 8-10) If a record is found for the url asked, a TCP connection is made to the **Network Stack**; including the necessary credentials for the request . Depending on the request, **Security Libraries** could be used on the request, to encapsulate data and send it securely.
—(Steps 11-12) When the **Network Stack** sends the request, if a response is received, this is given to the **Browser Window**.

—(Step 13) Wherever the response came from, the **Browser Window** will send the response to the **Resource Dispatcher**.

—The latter is in charge of sending the soon-to-be rendered page, but before doing it the dispatcher uses its **Message Filter** to send it to the right **Controlled Process**; this can be a **Web Content Renderer or WCR**, a **Plugin** or an **Extension**.

—(Steps 14-15) After the **Controlled Process** sends the bitmap for the resource obtained by the request from the **Browser User**, the **Browser Window** will save the rendered web page.

—(Steps 16-18) Finally, the **Browser Window** sends to the **UI** the bitmap to be shown by the display of the Browser User's machine.

2.4.7 *Alternative Flows*

—The resource obtained is not a web page, it could be a binary or a file. In this case, instead of rendering the response, the Web Browser only downloads it.

—The resource pointed by the URL does not exist.

—The request is cancelled by the user.

2.4.8 *Postconditions.* The Browser receives the resource indicated by the URL, which it is displayed by the peripheral device.

2.5 Implementation

—The Same Origin Policy (SOP) [Zalewsk 2008] is used to separate different resources by their domain, scheme and port. It is the minimum security mechanism a Web Browser has while requesting cross-origin resources, and divides the different kinds of contents so they cannot interfere with each other. To enforce the Same Origin Policy, Google Chrome, Firefox and Internet Explorer use different schemes [Barth et al. 2008; Barth et al. 2010; Grier et al. 2008; Reis and Gribble 2009; Silic et al. 2010; Vrbanec 2013; Microsoft ; Brinkmann 2015]; for example, Google Chrome leaves pages/resources isolated by creating for each content a **Rendering Engine**. Our solution abstracts this behavior with the **Resource Dispatcher** and the **Message Filter**, that sends the different requests from the **Web Browser Kernel or WBK** to the **Controlled Processes**, by using instantiations of **Web Content Renderer or WCR**s.

—The SSL/TLS protocol complements this pattern while providing security for communication channels between the Web Browser and provider. The **Security Library** in our pattern abstracts the security mechanism implemented on all Web Browsers.

2.6 Consequences

The Web Browser Kernel pattern provides the following benefits:

—**Separation of concerns:** Since the **WBK** executes the **main action flow** of a Web Browser, it would use the **Browser User**'s privileges to obtain the needed resources; we do not include actions related to the **Rendering Engine or RE**.

—**Security:** The **WBK** will only execute **Browser User**'s privileged actions when the Web Browser's **main action flow** needs to. Other types of action, like rendering resources (**rendering flow**), will have to pass a filter before; Traffic incoming from the **Controlled Processes** to the **WBK**, will be checked by the **Message Filter**.

—**Fined-Grained Control:** The **Resource Dispatcher** and **Message Filter** work together to send internal messages between the internal components of the Web Browser.

—**Performance:** Depending the architecture type selected **Monolithic Architecture** or **Modular Architecture**, the pattern can deliver different types of results. But, by separating the **main control flow** of the Web Browser from the **rendering flow**, it already improves the performance in either architectural types.

—**Reuse/Modularization:** By modularizing the **main control flow** of the Web Browser in the current pattern (and the same for the **rendering flow**), we can use our pattern to implement traditional and modern Web Browsers architetures.

This pattern has the following liabilities:

—Resources from providers/servers which do not comply with the specifications of the W3C, will be displayed incorrectly by the Web Browser.

—Session Cookies stored on the Browser Storage are the Achilles heel for the Web Browser. Even if they provide us with a stateful communication, they can affect tremendously the Browser User and the Provider on the Internet by becoming attack vectors for XSS or CSRF attacks, whenever a Web Browser has exploitable vulnerabilities or bad implemented functions [Sulatycki and Fernandez 2015].

2.7 Known Uses

—Google Chrome is based on a **Modular Architecture**, where a **Browser Engine** acts as the main process of its Chrome and Chromium (Open Source) Web Browsers[Barth et al. 2008].

—Internet Explorer and Edge are proprietary Web Browsers, which do not give much information about their structure or implementation. [Crowley 2010; Microsoft 2008] address the Loosely-Coupled architecture of Internet Explorer which is also a **Modular Architecture** and implements a central piece like a **Browser Engine** to work.

—Firefox, from the Mozilla Foundation, recently started rolling out its 52th version of its Web Browser with **Modular Architecture** fully implemented, though is still a work in process because it only creates 5 renderer instances by default. Mozilla brought the multiprocess Web Browser and sandboxing to Firefox which, on Windows, is based on the Chromium sandbox that Google uses in Chrome [Brinkmann 2015; Brinkmann 2016].

2.8 Related Patterns

—The **Web Browser Communication** pattern presents the components of the Web Browser and how they communicate with each other when a resource is requested [Silva et al. 2016b].

—The **Web Content Renderer** pattern describes the components of a web renderer of the Web Browser. It is in charge of composing and obtaining bitmaps of the requested web resources [Silva et al. 2016c].

—The **Reified Reference Monitor** [Fernandez 2013], describes how to enforce authorization rights when a subject requests access to a protected object or service and returns a decision (response).

—The **Cross-Site Scripting** attack [Sulatycki and Fernandez 2015] describes a common Web Browser attack, where server's vulnerabilities are subverted to attack the **Browser User**.

2.9 Conclusion

A Web Browser is a complex software and web application developers need to understand of how a Web Browser works, what components make this web client, their interactions inside the Web Browser, and the mechanism involved in: (1) the communication with the Web Server and (2) how a web page is rendered. In the current paper, we described the central piece that controls a Web Browser's **main action flow**: the **Web Browser Kernel**; which is the abstraction of the **Browser Engine** implemented in almost all types of existent Web Browsers.

Our aim with this and our previous work is to make understandable the internals of the Web Browser and the other mechanisms by using architectural patterns to construct a Reference architecture (RA) for Web Browsers. Our RA has been formulated by combining the **Web Browser Communication** pattern, the **Web Content Renderer** and now the **Web Browser Kernel**. These three patterns abstract the infrastructure of a Web Browser to help others understand holistically the components, interactions and relationships of this system.

2.10 Acknowledgements

We thank our shepherd Allen Wirfs-Brock and the "Strong Centers Group" conformed by Peng Zhang, Sumit Kalra, Lukas Reinfurt, Eduardo Guerra and Jiwon Kim, for the useful comments that significantly improved the quality of the paper. We also thank Y C Cheng for supervising our paper shepherding.

REFERENCES

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

Samuil Angelov, Paul Grefen, and Danny Greefhorst. 2012. A framework for analysis and design of software reference architectures. *Information and Software Technology* 54, 4 (April 2012), 417–431. DOI:http://dx.doi.org/10.1016/j.infsof.2011.11.009

Paris Avgeriou. 2003. Describing, Instantiating and Evaluating a Reference Architecture: A Case Study. *Enterprise Architect Journal* 342, 1 (2003), 347. DOI:http://dx.doi.org/10.1097/MAJ.0b013e3182314ba8

Adam Barth, Adrienne Porter Felt, Prateek Saxena, and Aaron Boodman. 2010. Protecting Browsers from Extension Vulnerabilities. *Ndss* 147 (2010), 1315–1329. DOI:http://dx.doi.org/10.1111/j.1365-2486.2006.01169.x

Adam Barth, Collin Jackson, Charles Reis, TGC Team, and others. 2008. The Security Architecture of the Chromium browser. (2008).

Martin Brinkmann. 2015. The state of multi-process architecture in Firefox. (2015). https://www.ghacks.net/2015/05/02/the-state-of-multi-process-architecture-in-firefox/

Martin Brinkmann. 2016. Multi-Process Firefox: everything you need to know. (2016). https://www.ghacks.net/2016/07/22/multi-process-firefox/

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

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

Eduardo B. Fernandez. 2013. *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons.

Eduardo B. Fernandez, Raul Monge, and Keiko Hashizume. 2016. Building a Security Reference Architecture for Cloud Systems. *Requir. Eng.* 21, 2 (June 2016), 225–249. DOI:http://dx.doi.org/10.1007/s00766-014-0218-7

Matthias Galster and Paris Avgeriou. 2011. Empirically-grounded Reference Architectures: A Proposal. (2011), 153–157.

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1994. *Design patterns: elements of reusable object-oriented software*. Pearson Education.

C. Grier, Shuo Tang Shuo Tang, and S.T. T King. 2008. Secure Web Browsing with the OP Web Browser. *2008 IEEE Symposium on Security and Privacy (sp 2008)* Sp (2008), 402–416. DOI:http://dx.doi.org/10.1109/SP.2008.19

Microsoft. Internet Explorer Architecture (Internet Explorer). (????). Retrieved 2015-09-17 from https://msdn.microsoft.com/en-us/library/aa741312(v=vs.85).aspx

Microsoft. 2008. IE8 and Loosely-Coupled IE (LCIE) - IEBlog - Site Home. (2008).

Charles Reis and Steven D Gribble. 2009. Isolating web programs in modern browser architectures. *Proceedings of the fourth ACM european conference on Computer systems EuroSys 09* 25, 1 (2009), 219. DOI:http://dx.doi.org/10.1145/1519065.1519090

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

Paulina Silva, Raúl Monge, and Eduardo B. Fernandez. 2016a. A Misuse Pattern for Web Browsers: Interception of traffic. *Proceedings of the 5th Asian Conference on Pattern Languages of Programs (AsianPLoP) 2016, Taipei, Taiwan* (2016).

Paulina Silva, Raúl Monge, and Eduardo B. Fernandez. 2016b. A Reference Architecture for web browsers: Part I, A pattern for Web Browser Communication. *Proceedings of the 5th Asian Conference on Pattern Languages of Programs (AsianPLoP) 2016, Taipei, Taiwan* (2016).

Paulina Silva, Raúl Monge, and Eduardo B. Fernandez. 2016c. A Reference Architecture for Web Browsers: Part II, a Pattern for Web Browser Content Renderer. In *Proceedings of the 21st European Conference on Pattern Languages of Programs (EuroPlop '16)*. ACM, New York, NY, USA, Article 27, 10 pages. DOI:http://dx.doi.org/10.1145/3011784.3011813

Rohini Sulatycki and Eduardo B. Fernandez. 2015. A Threat Pattern for the "Cross-site Scripting (XSS)" Attack. In *Proceedings of the 22Nd Conference on Pattern Languages of Programs (PLoP '15)*. The Hillside Group, USA, Article 16, 9 pages.

Tedo Vrbanec. 2013. The evolution of web browser architecture. (2013), 472–480.

Xin Wu. 2014. Secure browser architecture based on hardware virtualization. *International Conference on Advanced Communication Technology, ICACT* (2014), 489–495. DOI:http://dx.doi.org/10.1109/ICACT.2014.6779009

Michal Zalewsk. 2008. Browser Security Handbook, part 2. (2008).