SafeMash

A Platform for Safety Mashup Composition

Carlo Marcelo Revoredo da Silva, Ricardo Batista Rodrigues, Rafael Roque de Souza and Vinicius Cardoso Garcia
Informatics Center, CIn, Federal University of Pernambuco, UFPE, Av. Jornalista Aníbal Fernandes, s/n, Cidade Universitária, Recife, Brazil
{cmrs, rbr, rss4, vcg}@cin.ufpe.br

Keywords: Web Mashups, Security Policies, Information Flow.

Abstract: This article describes the SafeMash project, a platform that provides an environment for the construction, safe consumption and standardized of Mashups. The platform proposal is to offer functionalities focused in security aspects regarding the integration between web applications, the users and third parties APIs. Which is based in one specification to build an standardized Mashup. Those resources are based in security approaches specified by organizations such as OWASP and OpenMashup Alliance.

1 INTRODUCTION

With the advent of the Web 2.0, the content produced and published was no longer for only reading: now the users interact sharing information on sites and services that make the data available, making the web programmable, allowing that computers and humans work on a cooperative way (Governor et al., 2009).

In this context, arises the Mashups, applications able to combine existing functionalities, data and interfaces in order to reach a more complex goal. The data from heterogeneous sources are integrated, normally available through Web Services. The result of the integration of all service invocations may, for example, be presented in a site with usability resources, through technologies such as the Extensible Markup Language (XML) and Asynchronous Javascript and XML (Ajax) (Bozzon et al., 2009).

On the other hand, one of the obstacles is the lack of standardization, besides; problems related with security must also be highlighted. According to (Allen, 2001), we may relate Information Security with the protection of the set of information aimed to one organization or user, based in characteristics of Confidentiality, Integrity and Availability (CIA), being applied to all aspects of information and data protection.

Also according to (Allen, 2001), a Security Policy represents a set of controls established and that must not be violated, in order to minimize vulnerabilities in APIs, preventing them from being exploited by attacks such as the Cross Site Script (XSS), which is defined as the possibility of malicious users to inject scripts in the client side, with huge consequences, such as for example changes on the graphic interface of the application, compromising the integrity of the content displayed (OWASP: XSS, 2013).

According to a group of researchers from the University of California at Berkeley, the lack of standardization of APIs is among the ten biggest obstacles to the adoption of Cloud Computing solutions (CN) (Armbrust et al., 2009).

Each Content Provider implements the API in its own way, causing problems of standardization and divergence between the security offered and desired. With this, the interfaces present with heterogeneity, which hinders the insertion of a Security Policy efficiently, making your users feel afraid to enter their information, to the extent that the data traffic are becoming increasingly valuable.

This article has the goal to describe a platform called SafeMash, presented as an architectural model that may be applied in any domain that wishes to execute Mashup compositions. The proposal is to make available an environment for three main objectives: (i) sanitize input and output data between parties involved at Mashup consumption, (ii) where users can build Mashup with standard specification,
and (iii) analyse the compliance of security policies intended by Mashup services.

This work is presented with the following structure: in the section 2, security questions in a standard Mashup environment are described; in the section 3 the proposal of our methodology is presented, and partial result of an objective of our proposal. Finally, in the section 4 final conclusions and future goals are described.

2 MASHUP ECOSYSTEM AND SECURITY ISSUES

The work is focused in the communication flow between Mashups and other parties involved. For a better understanding of the problem, this information flow will be described, in this section, being represented as an ecosystem. And finally, the goal is to research some relevant aspects about information security in those scenarios.

2.1 The Mashup Ecosystem

The goal of Mashups is to collect several data through one or more APIs, which are inherent to third party services, known as Content Providers (CP). Those data will be transformed in a combined content, which will be presented as an answer to the Mashup Consumer (MC) requisition. As an example scenario, the MC may be a web application that requires content through a specification made available by the Mashup site and waits, as an answer, a graphic, functional component, adaptable to its graphic interface.

Therefore, being able to interact with user and application, this kind of component is called a Widget (Wilson et al., 2012). In Figure 1, a workflow of a Mashup ecosystem is illustrated, where the application makes a requisition and receives, as the answer, a Widget.

2.1.1 The Consumer

For a better understanding, consider that the web application in the example is a webmail, that has in its first page a widget that presents itself to the user as a calendar, coming from an external service called MashCalendar, which is used by the user to manage its appointments.

The webmail makes available a section where the user configures the widget, informing its access credentials to the MashCalendar service. After confirming the sending of information, the user is redirected for the main page. It is in this moment that the webmail starts its role as a Mashup consumer.

2.1.2 The Intermediate

Following the flow, the webmail will communicate through a specification based interface (Mashup specification), previously agreed by the managers of the MashCalendar service and will demand the calendar for the Mashup application of the MashCalendar, which will make requisitions with one or more CP distributed in the web.

For each CP, it will be an obligation of the Mashup application, that is, of MashCalendar managers, the implementation of the communication with each CP respective API.

2.1.3 The Third-Party

Each CP will answer its requisition with a given data. For example, the application Mashup will demand information of national or regional holidays, maps describing the location and transit on the user displacement to each scheduled meeting, among other information. Because of that, the Mashup site will combine those answers, generating this way the content that the webmail application waits as an answer, which will result the calendar with the appointments scheduled by the user.

2.2 Security Issues

Based in this example is possible to consider some worrying aspects in relation to Security. The first is when the user gives to the webmail his MashCalendar credentials, which will need to make available for the webmail a specification that allows reliability and integrity of the information that will flow.

An important aspect is related to the answer that will be generated by MashCalendar, the webmail must be sure that it will not bring information and behaviors that might harm the user. Such as XSS or other attacks such as: Cross Site Request Forgery (CSRF), which consists in inserting malicious requisitions in a browser session opened by the user, allowing it to be stolen, and in that way compromising its credential confidentiality (OWASP: CSRF, 2013). Another potential attack vector for environments with graphical user interface is an attack known as clickjacking, is when an attacker uses multiple transparent or opaque layers that overlap one form to trick the user had the intention of clicking a link or button that is overlaid.
With this, the attacker "hijacks" user click, redirecting it to another application or domain. Nothing prevents this technique is also used to hijack clicks of a digital keyboard, often used in internet banking (OWASP: Clickjacking, 2013).

These attack vectors are exploited vulnerabilities in browsers. Every browser has a security layer in its architecture, which is based on behavior policies. One of these is known as the same-origin policy, that it is a practice where it inhibits JavaScript (JS) code load HTML documents, forms or frames from other domains.

However, this does not apply to the loading of other JS code, potentially able to break this policy through mechanisms that use XMLHttpRequest (Barth et al., 2008), such as jQuery (JQuery, 2013), JSONP (JSONP, 2013) and YQL (YQL, 2013).

3 THE SafeMash PLATFORM

In order to reach the goals proposed in this work, one research was made in literature about the state of the art of techniques of vulnerability detection and web applications/services. Publications from Open Web Application Security Project (OWASP) (OWASP: Top Ten Project, 2013) and Open Mashup Alliance (OMA) (Open Mashup Alliance, 2013) are considered as the fundaments for this work and, in order to ease references, at this work will be named as specifying organizations (SO).

3.1 Users Scenarios

In Figure 2, based on the previous example, we present a workflow of same widget consumption, but running at SafeMash platform. Regarding the utilization of SafeMash, 3 user scenarios must be considered: Consumer, Developer and Administrator, they interact with 4 main components such as “Sanitization Filter”, “Integration Manager”, "Security Policy" and “Services Repository”, which relate between 14 main actions: “Request Content”, “Build Mashup”, “Search API/Mashup”, “Define Policies”, “Manage Mashup”, “Sanitize Request”, “Sanitize Response”, “Combines Content”, “Response Data”, “Integrate Services”, “Get Service Policies”, “Search Service”, “Process Response” and “Get Content” as illustrated in Figure 3.

3.1.1 The Consumer User

This scenario happens when one user, guest or registered, accesses the platform. In this case the user action will only have the intention to use a service or composition cataloged or built in the platform through the resource through the action “Request Content”. When the content is asked for, the action called “Search Service” will be used, which will search services cataloged or built in the platform.

The user request will be monitored by the component “Sanitization Filter”, where it will be analyzed with the goal to minimize attacks based in malicious injections and vulnerabilities. This layer will use a meta-model based on techniques used by the SO.

However, it is important to mention that, independently of the outlook presented, some user requests, be them in the Consumer, Developer or Administrator scenario, established by the platform itself, will be considered inexorable to be measured by this layer.

The next step will be to redirect the user request for the Mashups or services requested, where they will be analyzed by the component “Security Policies”, with the goal of verifying if the solicited CP is fulfilling the specified security policy.

This layer is also based on a meta-model based in the SO recommendations. And finally, the platform makes the action “Response Content”, to retrieve the required response in providers contents spread around the web.
3.1.2 The Developer User

This scenario happens when the platform is accessed by a registered user that wishes to create a Mashup. Available to the user is a control panel that offers basically three actions:

The first action allows the user to build its Mashup, where it is possible to insert syntaxes through the action named “Build Mashup”, making use of the component “Integration Manager”, with syntaxes according to the specification Enterprise Markup Mashup Language (EMML), created by the OMA, based in XML, with the intention to make the creation of Mashups more homogeneous.

The second action allows the user to search for available CPs through the action “Search API/Mashup”, where the user may perform a semantic filtering and search for contents, catalogued in the component “Services Repository”, that it wishes to use to compose its Mashup. These are then interpreted into action “Integrate Services”. In this stage, it is also possible to make restrictions in which the user will specify which security resources might be considered. From this, the platform will use the component “Security Policy” in order to filter by Mashups or APIs that fulfill the specified resources. The third action allows the user to attribute a set of policies in its own Mashups, through the action “Define Policies”. In this context, it will specify all security resources applied to its composition. This information will later be analyzed and tested by the platform administrators.

3.1.3 The Administrator User

An administrator is a user with privileges to perform an analysis through the action “Manage Mashup” in each composition created by a Mashup developer. From that analysis, the proposed composition will be released or not to the public, through the component “Security Policies”. This practice will contribute to other users, that in the future will consume, to have knowledge about
which security aspects are being considered by it, thus characterizing the environment of continuous security and standardization control.

3.2 Proposal Goals

In this section, we present some practices applied on the platform, which are directed to reach 3 desired goals by our proposal.

3.2.1 Sanitize Input/Output Flow

As a practical approach, we use a development release of our component "Sanitization Filter", in order to observe the impacts to be considered when it is inserted in real environments.

We develop a simple mashup that consumes a service through the Yahoo! Weather API (Yahoo! Weather, 2013), with parameter WOEID assuming the value 26802884 (City of Recife), where our component, through action "Sanitize Response," do a read from a byte array that represents the content as response.

Upon receiving this response, called the Mashup our component that makes a filter removing possible threats in response, for example, XSS, CSRF and Clickjacking techniques as described in Table 1, and Table 2 is an example of XSS attack using one of the techniques mentioned above.

Table 1: Some techniques applied to detect vulnerabilities.

<table>
<thead>
<tr>
<th>Threat</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSS</td>
<td>Malicious JS codes and tags; suspect code at JSON data blocks; HTML and URI encodeds; Inadequate uses of eval() or JSON.parse() functions;</td>
</tr>
<tr>
<td>Clickjacking</td>
<td>Malicious embed JS; Overlap suspect elements</td>
</tr>
<tr>
<td>CSRF</td>
<td>HTTP headers suspects; Tags IMG with suspect values in &quot;src&quot; attribute.</td>
</tr>
</tbody>
</table>

Table 2: An example of XSS attack using HTML/HEX encoding.

```
<script>alert('XSS');</script>
```

3.2.2 Standardize Composition

The Platform users have at their disposal a set of tools, such as drag and drop elements, a declarative language, besides a wizard composition, which will be responsible for any abstract complexity in the development of a Mashup, through action "integrate Services ", and consumption, through action" Combines Content ".

All content of the presentation layer will be transformed into EMML, bringing an oblique language between the control layer and presentation platform. The component "Specification Notation" will be responsible for performing validations syntax and operations EMML conversions in the code, resulting in a more specific language for the domain logic platform.

3.2.3 Promote Security Policies

The users themselves will describe what type of security practices should engage external services, such as the developer can define, through action "Define Policies" which a particular content of your Mashup can only be consumed from external service to perform the traffic content through https, or Customer may request, through the action "Get service Policies", the content at issue must be from an external service that has a well-structured policy which is committed to ensuring privacy in data traffic.

All these settings users will be managed by the component "Security Policy", and will be periodically evaluated by the Administrators of the platform, through the action "Manage Mashup".

4 CONCLUSIONS

In this article we present an environment ecosystem Mashups, describing benefits and purposes, as well as their weaknesses regarding security issues and the lack of standardization in these environments.

As proposal, we present a platform with the objective of minimizing the obstacles addressed. Currently, the proposal is in the implementation stage of its main components and documentation of architectural decisions.

As future works, the intention is to present the main non-functional requirements that must be fulfilled by the platform, in order to define the main functionalities. For the formal documentation the intention is to have Use Case Diagrams, which according to (Fowler, 2003), describe functionalities through diagrams with Unified Modeling Language (UML) notation, and that has the objective to elaborate a documentation describing functionalities in a graphic and intuitive way.
According an example in Figure 4, which describes the actions of Consumer, Developer and Administrator users, and the Behavior of the “Sanitization Layer” and “Policy Layer” components in the SafeMash Platform.

![Figure 4: UML Use Case Diagram of SafeMash Actors.](image)

Additional, we are developing two artifacts in low-level details: one which documents the main components and their relationships in the architecture, and one which specifies the layered architecture known as Layered View, which according to (Bachmann et al., 2001) provides greater flexibility in development since the architecture subdivides into distinct layers, facilitating the identification of the main features of the system and assists in the practice of reuse.

And we intend to present experimentations of the main components of the platform in real environments, using formal techniques to metric in an experiment in order to obtain satisfactory results in our goals.

REFERENCES


Open Mashup Alliance EMML Documentation, http://www.openmashup.org/omadocs/v1.0/, last access in April 2013.


OWASP: Cross Site Script (XSS), https://www.owasp.org/index.php/Crosssite_Scripting_(XSS), last access in April 2013.


