

Helsinki University of Technology  
Department of Media Technology  
T-111.5550 Seminar on Multimedia  
Spring 2008

11.04.2008

# Mashup Security

Jyrki Hakkola  
67799J

# Mashup Security

Jyrki Hakkola  
HUT, Department of Media Technology

`jyrki.hakkola@tkk.fi`

## Abstract

This paper considers the issue of mashup security. The most important technologies used in creating mashups, like Ajax, and the basic functionality behind the mashups are introduced shortly. After that the security issues concerning the technologies, the principles of mashups and the current security model of web browsers are discussed. The Same-Origin Policy which is the current security model implemented by browsers seems to be a source of many problems that mashups along with other Web 2.0 applications face today.

It can also be said that there are two kinds of security problems with the mashups: the problems that arise directly from the lacks of technology and hostile abuse of them and in other hand the problems that arise from the questions about trustworthiness of content. The latter one is more a problem of principle and it exists because of the nature of mashups.

The solutions for current problems and some examples of real security issues that have happened are introduced. Security is also a key issue when talking about widespread usage of mashups in enterprise. Some of the possible future scenarios of using enterprise mashups are discussed. Finally proposals for improving security and conclusions of current situation are made.

## 1 Introduction

This paper is made for course T-111.5550 Seminar on Multimedia organized by the Department of Media Technology at Helsinki University of Technology (HUT) at spring 2008. The paper considers the issue of mashup security.

Mashup is a web application that gathers data from several sources around the web, parses it and displays the result to user [1, 3]. The basic idea is to use information sources provided by other parties and use them together. The result has more value for user than single pieces of information. A traditional example could be that a map is loaded from one source and location information of some kind of services from another source. Mashup application could then display a map with the services placed on it as in Figure 1. The services could be anything from restaurants to houses that are going to be sold.

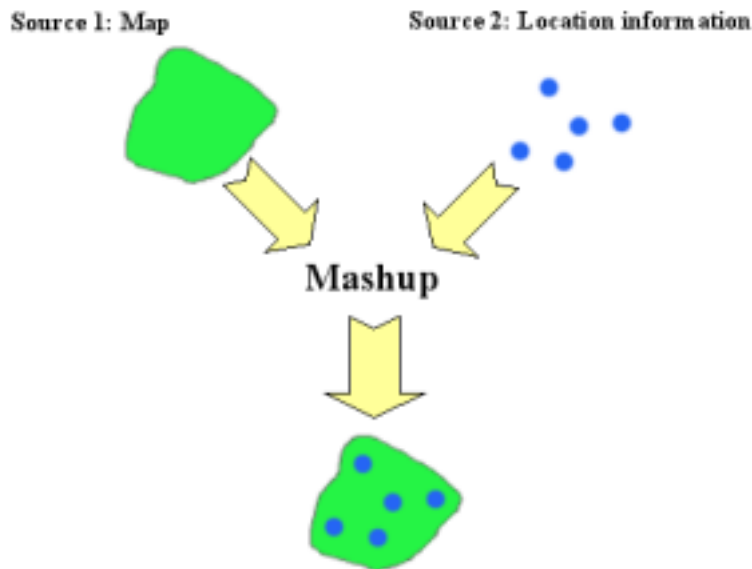


Figure 1: Example of mashup

Many providers are offering Application Programming Interfaces (API) which make accessing to services data sources easy. A good example is Google. Also different widgets that can be placed on web pages and which display some information from a service are widely used. And the amount is growing day by day. Today there are for example over 2800 mashups listed in <http://www.programmableweb.com> [11] which is a site that collects information about mashups and mashup technologies.

The next step in using mashups may be so called enterprise mashups [10, 15]. Interesting approaches could be for example mashing up internal data with public information like displaying shipping information of products on a Google map. Also different parties in product chain or different departments of company could benefit of easy access to commonly needed information. New business models like Software as a Service (SaaS) and Service Oriented Architecture (SOA) are rising in future. However in the enterprise business trust and security are even more critical issues than with non-business use. There are also usually strict rules of how company's internal information can be used and often public access to data cannot be allowed. Therefore it is critical to answer the questions about new kind of security models before there can be a massive breakthrough among enterprise.

Web browser's role is becoming more important day by day. It is not anymore just a tool for accessing static HTML-pages but a today's platform for Rich Internet Applications (RIA). Web 2.0 and mashups make browsing experience more and more like using a traditional desktop application. The traditional security model of browser is still quite complicated when talking about these new kind of applications. It can be said that there is either no trust between parties or then there is full trust [13]. This leads to dilemma that you can't have both, security and functionality [3, 13]. Always either one suffers. It is clear that new kind of security models are needed or the current ones have to be developed.

In this paper I will explain what kind of technologies are used to create mashups and how the current approach is vulnerable to attacks. I also introduce some proposals that have been made for improving the situation and finally make conclusions of situation.

Table 1: Examples of Same-Origin Policy

Address 1	Address 2	Same origin
http://a.domain.com	http://b.domain.com	No
http://a.domain.com/directory1	http://a.domain.com/directory2	Yes

## 2 Key technologies and functionality

Traditionally a web page was loaded once and when the content of page was to be updated the whole page had to be loaded again. Asynchronous JavaScript + XML (Ajax) is a technology that allows content being updated asynchronously without loading the whole page. XmlHttpRequest is an API that enables connections to remote sources via HTTP from client side. It is a key part of Ajax. Usually the transfer format used is XML, JavaScript Object Notation (JSON), HTML or plain text. In Ajax-applications JavaScript is used on client side to make connections and gather data by XmlHttpRequest and then to modify the page by accessing the Document Object Model (DOM) and CSS stylesheets of the page [1,3]. By using Ajax the data can be exchanged in small amounts with server and the user is provided an experience more and more like a desktop application without loading the whole page again and again. These kind of RIAs seem to be an essential part of growing amount of Web 2.0 applications [2]. Other technologies widely used in Web 2.0 applications are for example Really Simple Syndication (RSS).

The security issues of mashups focus on client side and usage of JavaScript and Ajax. The most typical problems are discussed further in the next section. Server side technologies are not considered in this paper. In Web 2.0 applications the content and scripts are gathered from many different sources. In server side final result cannot be affected too much. Instead it can be accessed only by the browser after all the pieces are put together. This makes the browser and things happening on client side to have a vital role in mashup security.

## 3 Security issues

The security issues with mashups are divided roughly into two categories in this paper: the security of technologies or practices used and the trustworthiness of content. The first category contains the issues that arise directly from the security problems and loop-holes of the technology used. The other category contains the issues that are not related to technology or hostile user behaviour but arise from the nature of mashups and are more of principal.

### 3.1 Current security model

The base of current browser security is the Same-Origin Policy (SOP). The idea is that by definition it is not safe enough to trust the content from other sites. The browser isolates documents from different domains from each other into a kind of a sandbox. Document from a domain cannot access content of documents in another browser windows or frames (content in <frame> or <iframe> tags) if they are from different domain. Different domains means that protocol, host or port is different from other domain. Subdirectories in a domain are of same origin although in practice they often belong to different owners. Table 1 shows some examples

of SOP. Resource files like scripts and images in a document are treated as components of the main document and therefore of same origin even if they existed in different domains. If documents are from same origin they can access each other's DOM and cookies and create connections to domain of document by using XMLHttpRequest.

There are still some ways to avoid SOP. They are using Ajax proxies, dynamic script tags and browser extensions and plugins [1]. Proxy server is from same origin than the document but forwards requests to third party services without of browser knowing it. Dynamic script tag means using the <script> -tag's src-parameter as a dynamic url possibly with parameters that make the requested address to return a specific content. These tags are exception of SOP and the content loaded by tag will handled as from the same origin than other elements in document. And then there are also extensions to browsers that allow using different functionality than was originally planned when creating SOP.

The nature of mashups is to use content from different sources. The nature of SOP is not to allow using content from different sources. This leads to earlier mentioned situation of all-or-nothing approach to security. If versatile and scalable functionality is wanted security often has to be sacrificed [3, 6, 13]. If SOP is avoided and access is given to a third party component it will right away have access to everything.

## **3.2 Technology and practices**

Some common vulnerabilities that web applications suffer are introduced in this section along with typical attack methods against them.

### **3.2.1 Cross-Site Scripting (XSS)**

Cross-Site Scripting is a type of attack where a trusted content is injected with malicious code. It is probably the most common and most dangerous type of attack concerning Web 2.0 services. XSS attacks can be used for example to steal session cookies, access restricted information, rewrite parts of the page or even act as a user of the browser. There are two types of this attack: reflected XSS and stored XSS [1]. The first one means that user is for example lured to click a link that sends JavaScript code to a trusted server as search parameters. The server reflects the code back as content of page by for example displaying the search parameters just used. The second type means for example that attacker inserts malicious JavaScript code into a guestbook, a discussion forum or some other service in web that allows user's to create content. In both cases a trusted page is somehow made to display malicious code which is then executed by the browser.

The execution of scripts can be so dangerous because SOP allows them to access everything in same origin. By using malicious code that runs instantly after loaded it is very easy to make harm. In Figure 2 there is an example of cookie theft introduced in [1]. At this example browser runs the code instantly and replaces the source parameter of first image in document with dynamic address to a evil domain. The address is generated by JavaScript and it contains cookie information which can be accessed because the script is part of the document itself and therefore of same origin. Request will be sent because sources of images are also handled as being in same domain as main document in SOP.

```
http://trusted.com/search?keyword=<script>
document.images[0].src="http://evil.com/steal?cookie=
"+ document.cookie; </script>
```

Figure 2: Example of stealing session cookies with JavaScript from [1]

The usage of Ajax makes the execution to happen in background outside the browser. This makes it even harder for user to notice that something unwanted is happening [7]. There are at least two well known examples of that: Samy and Yamanner.

At 2005 a guy called Samy created a worm in MySpace [12]. By default it was allowed to use only a few HTML tags and not JavaScript at all in MySpace. However by using certain tricks in constructing the content of the page Samy managed to bypass the input sanitization and was able to add active content to page. When someone was viewing his profile the script was executed and acted as a user currently viewing the page. It used Ajax to add Samy as a friend and a hero of the user. It also added the script itself to the victims profile so eventually everyone who viewed their profiles got infected too. The so called worm was spreading itself very effectively. This was noticed soon and for now same kind of situation cannot be created any more.

Yamanner was a worm in Yahoo! Mail at 2006 which also used XSS and Ajax [4, 7]. The neutralization of active content in mails failed and when a mail with certain malicious code was opened the code was executed by browser. The worm was able to act as a logged in user in Yahoo! domain and it used Ajax to search for e-mail addresses and to send mails to them. This is also fixed for now.

These two examples show how dangerous XSS can be. The dynamically created Web 2.0 pages create a great opportunity to this kind of attacks [1, 7]. The trend is that users create content and Web 2.0 applications and services are often intended to use rich user input. In the other hand the only way to avoid XSS attacks seems to be proper input validation and sanitization [1]. Here again comes the earlier mentioned problem of functionality versus security which the developes have to struggle with. If user's are going to be allowed to create versatile and dynamic content the security is often threatened in today's security model.

### 3.2.2 Cross-Site Request Forgeries (CSRF)

In web applications the authentication is usually carried out by using session cookies or HTTP authentication. The purpose is that the service kind of remembers that user is logged in and he does not need to enter username and password every time the page is loaded. It can be said that the web site trusts the user. Cross-Site Request Forgery is a type of attack that abuses this trust [1, 7]. The attack happens when a person is logged in a trusted service and visits other site that contains malicious code. When the code executes it can open a connection to the trusted service. The service allows connection because it comes from person's computer and it seems that the logged in user is opening the connection. The attacker can act as the trusted user of the computer and for example use online banking system, order stuff from a web store, send e-mail, access restricted information etc.

### 3.2.3 RSS Injection

RSS injection is a type of attack where the RSS feed is injected with malicious code. If the RSS reader used can display rich content and run scripts the same problems arise that exist while using the web browser.

### 3.2.4 Denial of Service (DoS)

Denial of Service attack means that a service is drowned with false requests. The attacker sends so many requests to the service that processing them all takes too much time to answer the real requests in time. Executing malicious JavaScript code makes this kind of attack possible. The code can for example make requests in a loop to the target service of the attack [1].

### 3.2.5 Non-professional developers

Mashups are often referred as end user tools [14]. Most of the mashups in web today are truly made by non-professional end users. The problem is that non-professional developers probably don't have enough knowledge nor experience to take security issues into account [8]. Poorly implemented web applications provide a great opportunity for attackers. Providers of mashup APIs have also great responsibility. The development tools should be designed keeping in mind that the users may not have a slightest idea of what secure web application really means.

## 3.3 Trust and principles

When the technology is working as expected and no-one is trying to attack and steal information or insert malicious functionality there may still be problems. The basic idea of mashups is to use information sources provided by others [1, 3, 9]. This leads to problems that are not related to technology but to principles and questions about trustworthiness of other parties. In this case it is not considered any more if the content is secure. The question that needs to be answered is that how can it be known if the information provided by other parties is true or not. Unfortunately the answer is that there hardly is a way to know. There is no technical method to measure this kind of trust.

## 3.4 Enterprise mashups

Especially when talking about enterprise mashups security and trust are essential. Enterprise mashups can be seen as a great opportunity but also as a nightmare of IT department. Most of the companies have already some kind of existing guidelines for security. Exposing internal data in public web is in many cases prohibited. Although it could be interesting and useful to use both company's internal data and the public data sources that could be problematic and in many cases impossible. Usage of external scripts in a company's domain cannot probably be allowed in any case because of full access to parent documents that comes with current security model. It could still be possible to just fetch information from external sources into inside of company's firewalls and no internal data is sent outside. This seems to be the a secure option

```

<div principal='blog-body'>
  <b>Blog entries</b>
  <div principal='blog-entry'>
    today's entry
  </div>
  <div principal='blog-entry'>
    yesterday's entry
  </div>
</div>

```

Figure 3: Example of a blog with principal annotations from [8]

in theory but there are still some risks in it too. It has to be guaranteed that the external content used is totally secure or at least it can be sanitized to prevent XSS.

Other interesting thing about enterprise and mashups is developers of mashups. If the mashups are going to be accessing critical or confidential data the developers cannot be just anyone. More of this is discussed in section 4.

## 4 Future

As the main security problems with mashups seem to result from security models lagging behind the needs of today's web applications. The obvious solution seems to be improving the security model and there exist some proposals for that.

One interesting approach is presented by Helen J. Wang among others [13]. Instead of SOP they introduced a security policy called Verifiable-Origin Policy (VOP). In SOP documents cannot access content of other domains than their own and the content in a document is fully trusted. In VOP the document could request information also from other domains. The request target could check the requester and decide how to answer to request. They also introduce some new tags to be used in web pages like `<ServiceInstance>`, `<Sandbox>` and `<OpenSandbox>`. Web developers could easily use content from other parties but they could decide if content they intend to use is trusted and can have access to the original content or not. It's like trusting the content enough to use it but not enough to allow it to access own original content.

One another approach is Douglas Crockford's proposal for a new `<module>` tag [5]. The approach is somehow like the one in [13]. The `<module>` tag could use content from other origins than the original document and it could communicate with outer layer in JSON format.

Benjamin Livshits and Ulfar Erlingsson are proposing principals as an addition to SOP [8]. HTML elements would have a new attribute: `principal`. Only elements with same set and order of principals could access each other. Principals of parent elements and principals of elements themselves would define the order. With this kind of approach it would be easy to create restrictions by using different sets of principals. This could probably also be easy to implement in browsers and mashup creation frameworks and tools.

An example of principals is introduced in Figure 3 (originally from [8]). In this example there are two principals used: `blog-body` and `blog-entry`. Both blog entries could access each other

because they have the same set of principals in same order but they could not access the DOM.

As has been said the trend seems to be that enterprise mashups will play bigger role at future. Probably they will become more popular step by step. First thing for a company would be to create guidelines for using mashups in addition to current security guidelines. It could be defined which data sources can be used and what internal information is allowed to be accessed by mashups. Wild use and creation of mashups by random employees is still not expected. More probable solution would be that mashups are created by employees capable and authorized of doing it and after a it is created and accepted in company it will be released as one of companys internal tools that may be used alongside of others.

## 5 Conclusions

My original approach for this study were to process security issues from two perspectives introduced in section 3: the issues arising from technology and the issues arising from principles. Soon it became clear that the issues of principle are not discussed as widely as the technological problems. Actually hardly any information of these questions was available. This is unfortunate. When talking about mashup security it would have been interesting to process these questions because they are the ones that exist because of mashups are what they are. Technological problems exist because mashups are created using a certain technology. Technology used can and probably will change as time goes by. The reason for lacking discussion of principles is probably that there really is no clear answer for them. They are more like common issues that are related in relationships and trust between human beings. Eventually people just decide to trust others and to trust the services created by others and it is very hard to invent any technology that could be used to help in these kind of decisions. This kind of trust can only be earned with enough time and after knowing each other.

When talking about technologies the issues of mashup security usually fall in the same category than the common security issues of Web 2.0. The main source of problems is that the principles of web browser security model are lagging behind the development pace of current web applications. Same origin policy doesn't fit as such because the trend is to use several origins. And either to fully trust to content or not to trust at all doesn't work if multiple sources of active content is used same time. If the current security model is tried to stretch the security is often sacrificed same time. And if an application is totally secure it is most probably totally unusable in today's context.

It has become clear that browser has become a platform for Web 2.0 applications including mashups instead of just being a tool to display hypertext. As it have been addressed in this paper most of the problems exist because the browser were originally designed for different function that it has today. The original duty was to display static HTML pages, not to run multiple applications sending and fetching information on the fly and the security model has originally been designed accordingly. A good question that arises when looking at security issues there are is if the browser really is an optimum choice for this task. Although this has become the way to do things today and in the future also as it seems. It is easy to point at reasons but difficult to imagine what could be a better alternative or how the security model could be changed instantly.

It seems that something really needs to be changed if the mashups along with other web applications are going to be secure, versatile and able to allow rich user input same time. Now

web browser has only two answers to question if content or a script is trusted or not: "yes" or "no". There should be a way to say "maybe" or "enough to display but not enough to gain access to my data". With today's technology it is not possible. Fortunately there are still some interesting proposals for improving this situation and allowing better way to decide which elements in a web page can access others and which not.

In technology's point of view the mashup security can and probably will improve in future at least when talking about today's problems. Meanwhile technology is evolving constantly and new technologies will come. The battle against security issues in technology cannot be win for good because with new technologies there will be new kind of loop-holes and problems that need to be answered. But at least situation with issues that exist because of yesterday's technologies are today used in a way that they were not planned to be used will be better.

There is also lots of discussion about enterprise mashups and that security needs to be enhanced before they can be used more widely. Still the question about how exactly the security should be enhanced remains mostly unanswered. Improving browser technology and creating standards for making mashups secure way will probably make things better.

## References

- [1] OpenAjax Alliance. Ajax and mashup security. <http://www.openajax.org/whitepapers/Ajax%20and%20Mashup%20Security.php>, 2008.
- [2] OpenAjax Alliance. Introducing ajax and openajax. <http://www.openajax.org/whitepapers/Introducing%20Ajax%20and%20OpenAjax.php>, 2008.
- [3] Brent Ashley. Shaping the future of secure ajax mashups. <http://www-128.ibm.com/developerworks/library/x-securemashups/>, 03 Apr 2007.
- [4] Eric Chien. Malicious yahoooligans. <http://www.symantec.com/avcenter/reference/malicious.yahoooligans.pdf>, August 2006.
- [5] Douglas Crockford. The <module> tag: A proposed solution to the mashup security problem. <http://www.json.org/module.html>, 30 October 2006.
- [6] Collin Jackson and Helen J. Wang. Subspace: secure cross-domain communication for web mashups. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 611–620, New York, NY, USA, 2007. ACM.
- [7] G. Lawton. Web 2.0 creates security challenges. *Computer*, 40(10):13–16, Oct. 2007.
- [8] Benjamin Livshits and Úlfar Erlingsson. Using web application construction frameworks to protect against code injection attacks. In *PLAS '07: Proceedings of the 2007 workshop on Programming languages and analysis for security*, pages 95–104, New York, NY, USA, 2007. ACM.
- [9] Paul Marks. 'Mashup' websites are a hacker's dream come true. *New Scientist magazine*, page 28, 12 May 2006.
- [10] Chong Minsk, Goh, Siew Poh, Lee, Wei, He, Puay Siew, and Tan. Web 2.0 concepts and technologies for dynamic b2b integration. *Emerging Technologies & Factory Automation, 2007. ETFA. IEEE Conference on*, pages 315–321, 25-28 Sept. 2007.

- [11] Programmableweb. <http://www.programmableweb.com>, February 2008.
- [12] Samy. The Samy worm. <http://namb.la/popular/>, October 2005.
- [13] Helen J. Wang, Xiaofeng Fan, Jon Howell, and Collin Jackson. Protection and communication abstractions for web browsers in MashupOS. In *SOSP '07: Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, pages 1–16, New York, NY, USA, 2007. ACM.
- [14] Jeffrey Wong and Jason I. Hong. Making mashups with marmite: towards end-user programming for the web. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1435–1444, New York, NY, USA, 2007. ACM.
- [15] Joe Zou and Christopher J. Pavlovski. Towards accountable enterprise mashup services. *e-Business Engineering, 2007. ICEBE 2007. IEEE International Conference on*, pages 205–212, 24–26 Oct. 2007.