Questioning the security of cross-domain components of Web 2.0 Mashups

Panagiotis Papadopoulos

panpap@csd.uoc.gr

Computer Science Department
University of Crete

August 2011

Table of Contents

Table of Figures

Abstract

The more the web 2.0 evolves the more increases the need for applications that combine components from multiple content providers in the user's browser. However, the current browsers were not designed to support such applications in terms of security. It becomes obvious that the browser's insufficient design creates a security gap as these applications are concerned. This is because the information leakage that violates the privacy of any user is an important threat the web 2.0 applications face nowadays. This report investigates the web pages' content and especially cross-domain components and the Http requests that these components produce in a user's browser.

Keywords: Web 2.0, mashups, component model, browser, Http requests

1. Introduction

1.1. Web 2.0

The term Web 2.0 is associated with web applications that facilitate participatory information sharing, interoperability, user-centered design, and collaboration on the World Wide Web. In contrast to websites where users (consumers) are limited to the passive viewing of content that was created for them a Web 2.0 site allows users to interact and collaborate with each other in a social media dialogue as creators of user-generated content in a virtual community. Examples of Web 2.0 include social networking sites, blogs, wikis, video sharing sites, hosted services, web applications, mashups and folksonomies: [1]

Web 1.0		***Web 2.0***
DoubleClick	→	Google AdSense
Ofoto	→	Flickr
Akamai	→	BitTorrent
mp3.com	→	Napster
Britannica Online	→	Wikipedia
personal websites	→	blogging
evite	→	upcoming.org and EVDB
domain name speculation	→	search engine optimization
page views	→	cost per click
screen scraping	→	web services
publishing	→	participation
content management systems	→	wikis
directories (taxonomy)	→	tagging ("folksonomy")
stickiness	→	syndication

1.2. Web 2.0 Threats

The downside of Web 2.0 is that Web applications are vulnerable to both internal and external threats .More Internet attacks target Web applications than all other services combined. The sharp increases in SQL injection [2] and Cross Site Scripting (XSS) [3] as well as the emergency of threats such as the Cross Site Request Forgery (CSRF) [4] and the botnet attacks [5] provide hackers with a powerful arsenal to unleash on Web sites.

Web 2.0 is faced not only with possible security problems but also with privacy issues as in search engines that most people use to navigate the internet. Search engines have and make use of their ability to track each one of someone's searches. They can record user's IP address, the search terms he uses, the time he spends on his search, and other information. Internet privacy, in general, involves not only the exercise of control over the type and amount of information revealed about a person on the Internet but also who may access this information.

This report studies how components from cross-domain contents could communicate with their web servers by sending requests without the user's knowledge. Moreover this report discusses how that could become a serious threat for the user's web profile and his privacy in general. For that purpose we have created a system that analyzes the content of a Web 2.0 site by monitoring the http requests/responses a user's browser sends/receives.

Since content in a mashup application can stem from multiple untrusting providers; in order to control these mashup's components we have created a system that handles each component's Http requests. We believe that using our proposed system can help us control the information that these requests transfer towards the web. Control over the information is vital as a leakage could lead to the exposure of personal transactions thus violating the user's privacy or to the change of the personal transactions, which creates many possibilities in stigmatizing user's profile.

This report is divided into two parts. Part I describes the system that is responsible for analyzing the contents of a number of websites and Part II describes the mechanism which handles the Http requests that components produce.

2. Part I – Mashups' Content Analysis System

In recent years, the rapid spread of the use of scripting on the client side and of the use of scripting languages like AJAX [6] has helped mashup [7] Web pages to develop. These Mashup Web pages combine components from different content providers. Mashups are now prevalent in a number of contexts which have integrity requirements, or in contexts which handle confidential information. They are also necessary to an advertising supported business model, and they support user-generated content in Web 2.0 websites. Applications that use combined content are also widely used in contexts with stricter data security requirements, like consumer banking sites, on-line shops. Because the browsers' security models were defined and developed without anticipating such applications, these applications are barely supported by the security model the browser has to offer. So it is under discussion if the current browsers can fully support such applications.

This fact has intrigued us to investigate more thoroughly these mashup applications and even more the parts they consist of in the era of Web 2.0. This interest motivated us to create a system which would automatically analyze the most popular websites around the world and produce accurate information about their contents; more precisely the origin of these contents and their affiliation with the websites that contain them.

Our system's architecture consists of four main modules: the Controller, the Browser, the Observer and the Analyzer. This architecture is presented in more detail in the Figure below:

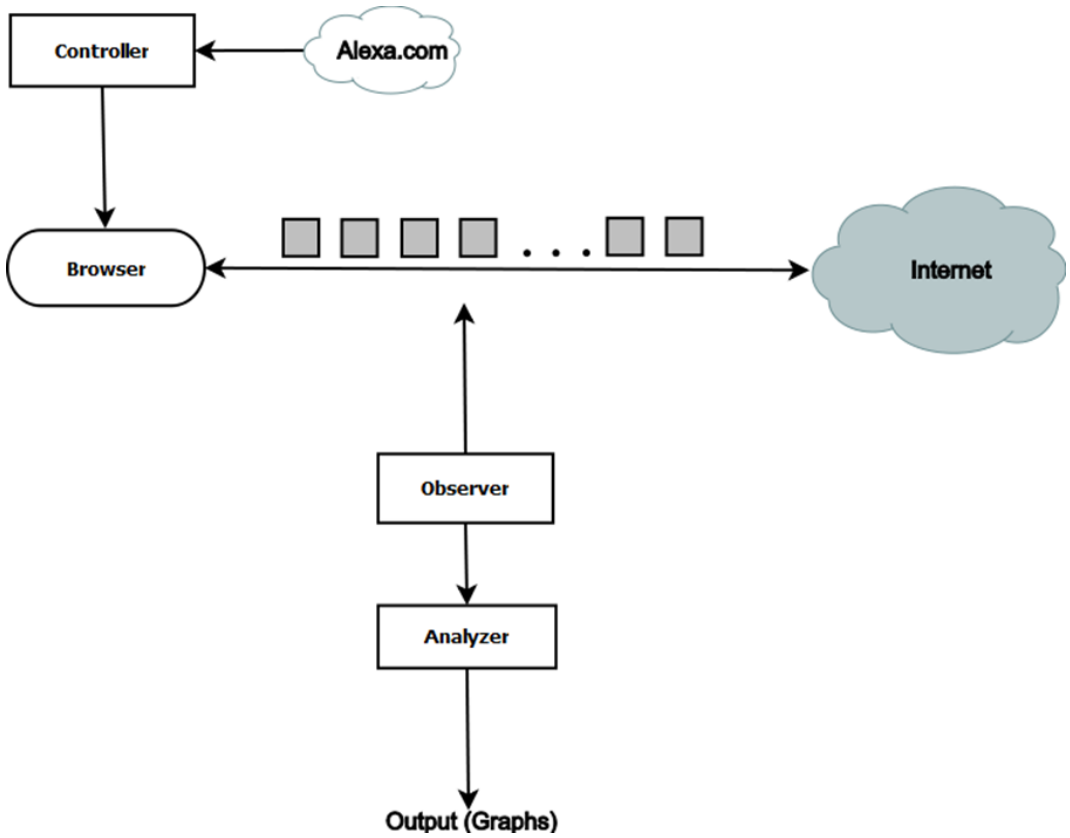


Figure 1*: Web site Content Analysis System's Architecture*

The functionality of each module is described below:

- The **Controller** checks the existence of a report from Alexa [8]. If this report does not exist, the controller will automatically download the latest update of the posted report, otherwise it will check if the report it already has is up-to-date. If it is not it will download the latest version. This function prevents the system from making unnecessary downloads and from exceeding overhead. Once the controller has obtained the updated report, depending on the parameter given as ingress to the system it will obtain the first N websites for analyzing, and he will send every one of these websites to the Browser in order to be fetched.

- The **Browser** is responsible for calling the web pages the Controller asks for, for a specific amount of time. A condition not to be violated is that the user should by no means interrupt the browser. The reason is that this system requires the content from

loading phase of a web page so this phase should not be distorted by any triggered by the user event. This is the reason why the browser application used is stripped from every personal settings, plug-ins, extensions, saved browsing history and active cookies which would affect the way that browser is fetching the web pages.

- The **Observer** is responsible for monitoring the Browser's activity and capturing the packets exchanged with the internet. In this way the system retains a wide picture of the different kinds of requests the browser is needed to send towards the internet in order for the browsing to be completed successfully for every website.

- The **Analyzer** is responsible for analyzing the results given by the Observer. Simply put, the Analyzer loads the data saved by the Observer and then highlights the information we seek from the requests that the Browser sends towards internet and also about the components the Browser asks from the server. In other words, the Analyzer initially retrieves the components every website uses. Then, for every component, the analyzer records not only its source address but also the number of the components the website receives from that source address. Afterwards, the analyzer investigates the ownership of every single one of these addresses, their origin and their relationship with the analyzed website. Finally, it divides them to three groups depending on the obtained information. These three groups will be described on the following sector.

2.1. Getting traces and parsing packets

As previously described, a basic function of the Analyzer module, and of our system in total, is to record and analyze the traffic created during the browsing of every webpage. This is achieved through a process that initially monitors the packages that are delivered to the Browser's port and continues with the filtering of these packages. Only packages that follow a specific protocol, the Hypertext Transfer Protocol (HTTP), are retained whereas all the other packages are discarded. This procedure is completed with the removal of the *query string* from the package's source address.

Query string which is the part between the "?" character and the end of the URI (or the character "#") passes to the application, and consists of one or more fields=value pairs, separated by either "&" or ";" character. For example at:
http://host/path/somepage.pl?name=john&age=32Parameter",
name=john&age=32Parameter" is the query string that will be removed.

2.2. Components' origin

Another basic function of the Analyzer module is to discover the source address of every object used by a website and also to whom this address belongs. This is easily accomplished by tracking the host from the URL that already exists in the source address of the cached packet. Afterwards, in order to find the owner of the address we have just obtained, we need information about the hosts and to whom (organization, company) each host belongs. This information is stored in the web and access to this information is gained by using the *WHOIS* protocol [9].

WHOIS is a query and response protocol that is widely used for querying databases that store the registered users or assignees of an Internet resource, such as a domain name, an IP address block, or an autonomous system. The protocol stores and delivers database content in a human-readable format.

Finally, we have classified every component into three categories depending on the "distance" of the origin of the address of the parental website. These categories are:

- The First class (inner domains), encloses all the components originated from the same parental website.

- The Second class (semi-outer domains), encloses all the components that belong to hosts with the same owner as the parental website we analyze, but have different host name from the owner. For example if "yahoo.com" is the parental website and the "l1.yimg.com" is the source address of the component we analyze, though WHOIS we can discover if these addresses belong to the same owner and if so, we classify this component to the Second class.

- The Third class (outer domains), contains all the components that belong to neither the two classes we have described above. In other words, the components enclosed to this class, belong to an outer owner and are just used from the parental website. For example if the parental website is the "imdb.com" and (the) "platform.twitter.com" is the source address of the component we analyze, we categorize this component as a third class since, the information provided from WHOIS proves that the component does belong to the same owner as the parental website.

3. Part II– Requests Marking System

During the construction of our system, the results of the experiments we viewed in Part I, have revealed that many popular websites embed third-party content in frames, relying on the browser's security policy to protect them from malicious content.

Frames, however, are often insufficient isolation primitives because most browsers let framed content to manipulate other frames through navigation. This fact led us to believe that these components had to be further inspected as their function, in many cases, could be characterized as malicious.

An example of an attack against the user, by exploiting the existence of these components in a rather safe website, could be the following: someone inserts an object in a safe website in a form that does not bring the slightest suspicion that could be vulnerable, such as an advertising banner. This object, at the background, sends periodically requests, which could be queries in a search engine, like Google or yahoo, with illegal content, such as child pornography, drug purchase or even instructions of manufacturing explosives, without the user realizing it. This results in polluting the user's account in the search machine, due to searches, which even though he has never made, his prints imply otherwise. Unfortunately, this process allows the existence of material and evidence able to accuse a user for actions he has never made in reality.

This scenario gave us the motive to create a system which would enable us to examine the kinds of requests that exit the user's browser without him realizing it. In the long run this system will help us identify similar cases to our previously mentioned scenario.

The system we have created for that purpose will be described in this part. The idea behind our proposed system is to find a way to control the requests sent by the browser towards the internet and separate them into two groups; those triggered by the user and those triggered by any other cause. The most effective way to control the requests was to mark those that were sent after the actions of the user, such as the movement of the mouse pointer, making a click with the mouse, or pressing a key from the keyboard. These actions make up the category of events.

In order to construct this system we have used an extension of a well-known browser. This extension is able to handle these events that are taking place at the browser and then mark the requests that are going to be originated from these events. Apart from the extension, a proxy server is needed to monitor the outgoing requests, and to establish a communication with the browser and the outside world.

The following Figure presents the architecture of the proposed system:

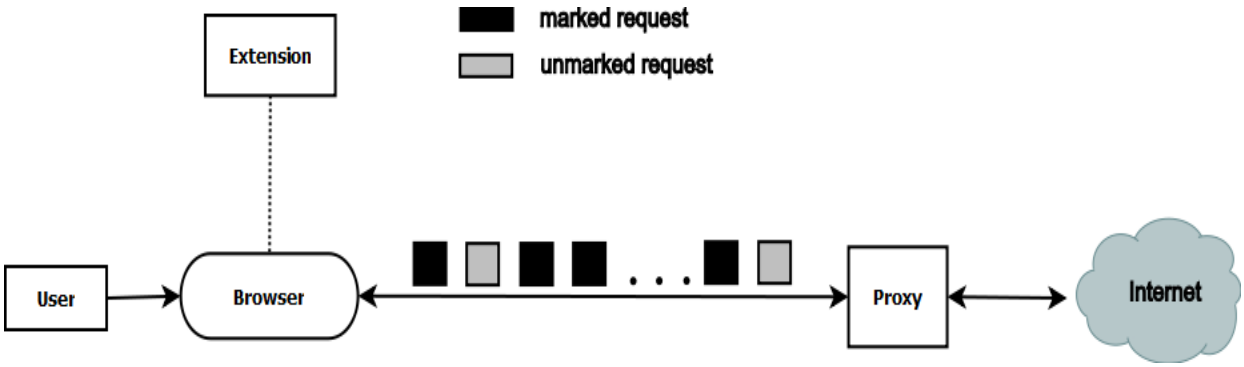


Figure 2: *Requests Marking System's Architecture*

It is now apparent that by simply monitoring the data flow that passes through the proxy server, we can easily separate the requests and afterwards analyze and identify the maliciousness of them.

3.1. Http Request/Response header

One of the basic components of our system, but also one of the fundamental components of the web's structure, is the HTTP protocol. HTTP is one of the core technologies behind the Web. In addition to the actual content, some important information is passed through HTTP headers for both HTTP requests and responses. They define the operating parameters of an HTTP transaction.

A Typical HTTP Request:

GET /servlet/Search?keywords=servlets+jsp HTTP/1.1
Accept: image/gif, image/jpg, */*
Accept-Encoding: gzip
Connection: Keep-Alive
Cookie: userID=id456578
Host: www.somebookstore.com
Referer: http://www.somebookstore.com/findbooks.html
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)

This header is of great importance to our system since it is used as a means of transferring the additional information we are interested in, in order to draw conclusions about our experiment.

3.2. *Event handling*

The most common programming language used for webpage construction today is JavaScript. JavaScript offers the developer the opportunity to create dynamic event-driven web pages. Every webpage is possible to have elements which create actions detectable by JavaScript. We have dealt extensively with events in our system so we will further explain their function. Every element on a web page has certain events which can trigger a JavaScript. For example, we can use the onClick event of a button element to indicate that a function will run when a user clicks on this button.

Examples of events we have dealt with in this work:

- A mouse click
- Mousing over a hot spot on the web page
- Selecting an input field in an HTML form
- Submitting an HTML form
- A keystroke

The following Figure describes in a very basic way how the various actors in the event model interact with one another.

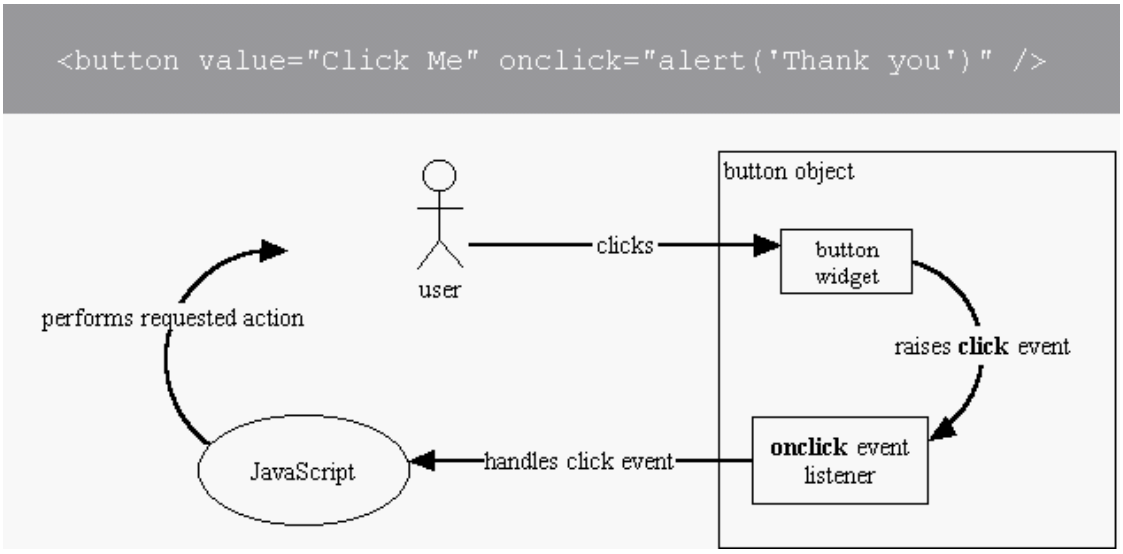


Figure 3*: The interaction in event model*

Event Bubbling

The availability of events in places other than their element of origin is known as “event propagation” or “event bubbling”. Event bubbling offers the ability to handle events anywhere above the event-raising element in the hierarchy.

For example, when one of the menuitems, visible in the following Figure, raises an event, any element above it in the hierarchy is able to handle it. More specifically, when the menuitem raises the "command" event, indicating that it has been selected, the menupopup, the menu itself, the parent box or the root window element itself can all take advantage of this.

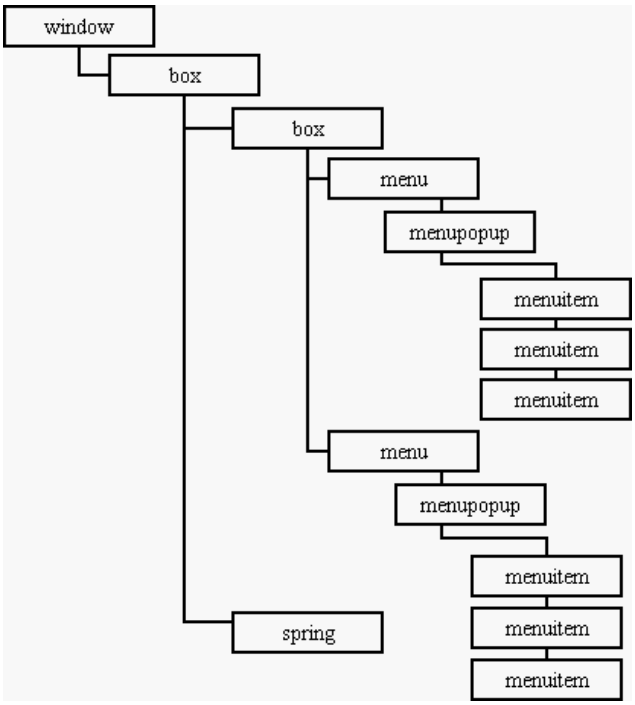


Figure 4*: Event bubbling*

Event Capturing

Event capturing is the complement of event bubbling. When someone registers an event listener on an ancestor of the target event (i.e. any node above the event raising element in the node hierarchy), they can use event capturing to handle the event in the ancestor before it is heard in the target itself or in any of the intervening nodes.

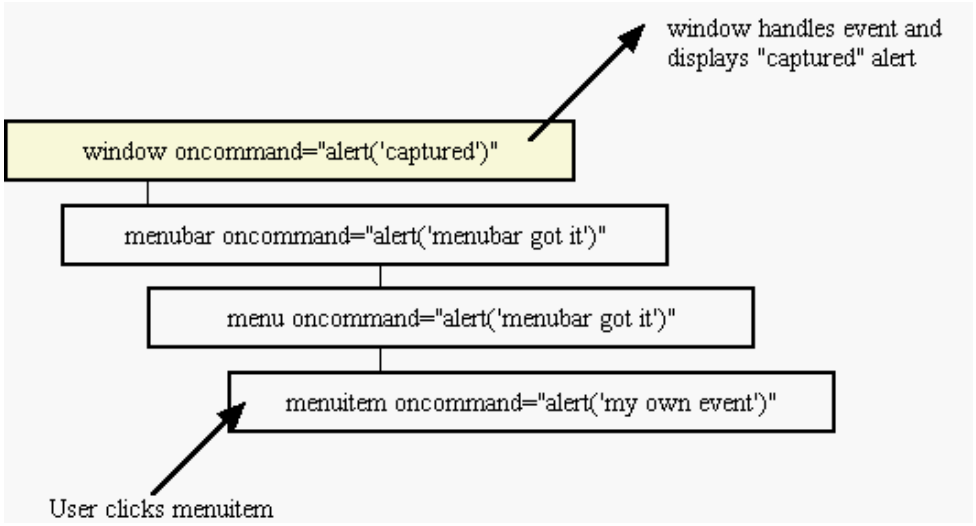


Figure 5*: Event capturing*

In Figure 5, the alert dialog invoked by the menuitem itself is not displayed, since the root window element has used event capture to handle the event itself.

It is more than obvious that event handling is very important to an event-driven webpage of a website. In our system we monitor through event listeners the events that appear in the browser and we record the cause of their creation. Thus, we are able to distinguish the events caused by the user, like onClik, OnMouseover, OnKeypress, from those caused without the user's interference, like time-triggered events.

3.3. Http Request Parameter addition

With the aim to monitor the events taking place inside the browser, we used the request header, and all the functions that were described in the sections above, in order to transfer information from the browser towards the Internet. This occurred with the addition of a parameter to the Http request header that the browser-client sends to the server. Every time an event is caught in the browser, it is the kind of the event (e.g. onClick, onKeypress, onMouseover) that is entered in this parameter and thus gives us the information at any time if this request was created from the user's action. In more detail, it was necessary to use an HttpChannel[i] in order to deal with HTTP requests and responses. In particular, to get the HttpChannel just before the HTTP request is made we need to observe the "http-on-modify-request" topic.

[i] An HttpChannel in Mozilla Firefox is a component implementing nsIHttpChannel interface..

We have also created an Observer[ii] to gain the ability to interfere to requests that have already been initiated. Once we had obtained the HttpChannel, we were able to connect with the Event Listener. After the Event listener has captured an event, the request header will be modified due to the addition of a parameter named "marked". This parameter contains the kind of the event, like onClick, onKeypress, onMouseOver.

So, we can easily distinguish, later, with the assistance of the proxy server, the requests that have passed through our system and have this extra parameter from those who have not passed and are "unmarked".

4. Implementation

Our system in part I initially used two parameters: N (the number of websites to be used from Alexa) and T (the number of seconds we need for a website to be fetched by our browser). Initially, the Controller will download the updated report of the "Top sites on the web" from Alexa.com. Then it will take the N first websites from the report and send these websites to the Browser (in this report we have used Mozilla Firefox) to fetch each one of them. Afterwards, the Observer, by using Tshark [10] network analyzer for a specific amount of time, will get each website's trace by monitoring the network traffic from the port Browser is using. Then the Analyzer will take the data the Observer has captured and stored and will use the Dpkt module [11] to parse the HTTP header from the packets that were captured. Finally, our system uses fwhois (a program written in C that sends queries to any Whois server) in order to discover the owner of the domain that is providing a component to the website we investigate.

It is important to mention that the Controller module was programmed in Ruby, the Analyzer module was programmed in Ruby, Python and bash script and finally we have used Jgraph [12] to automatically generate graphs from the system's output.

In part II, we have used a browser (Mozilla Firefox 4 with Gecko 2.0[13] engine which is written in C++ and is cross-platform) with an extension that we have implemented using JavaScript and XUL [14]. The XUL was used for implementing an elementary graphical interface. The browser is responsible for fetching the websites from Alexa and the extension is responsible for handling events triggered from the user and adding a parameter in the http header to mark the request for the Burpsuite [15] proxy, when an event has been caused by the user.

[ii] An observer in Mozilla Firefox is a component implementing nsIObserver interface.

5. Evaluation

5.1. Website Content Analysis System

To present an evaluation of the Content Analysis System we have used the first 100 websites from Alexa's "Top sites on the web" and we have fetched each website for 10 sec using a 24Mbps internet connection. For the purposes of our discussion we have used a sample of 5 of these 100 websites. Graphs from each of these 5 websites and a final graph for the whole number of the websites we have analyzed are presented.

Used:	100 of 1.000.000 Alexa's top websites
Browsing time per website:	10 sec
Internet connection:	24Mbps
Monitoring time:	18' and 44''
Analysis time:	37'' per website

Table 1: Centralized data table

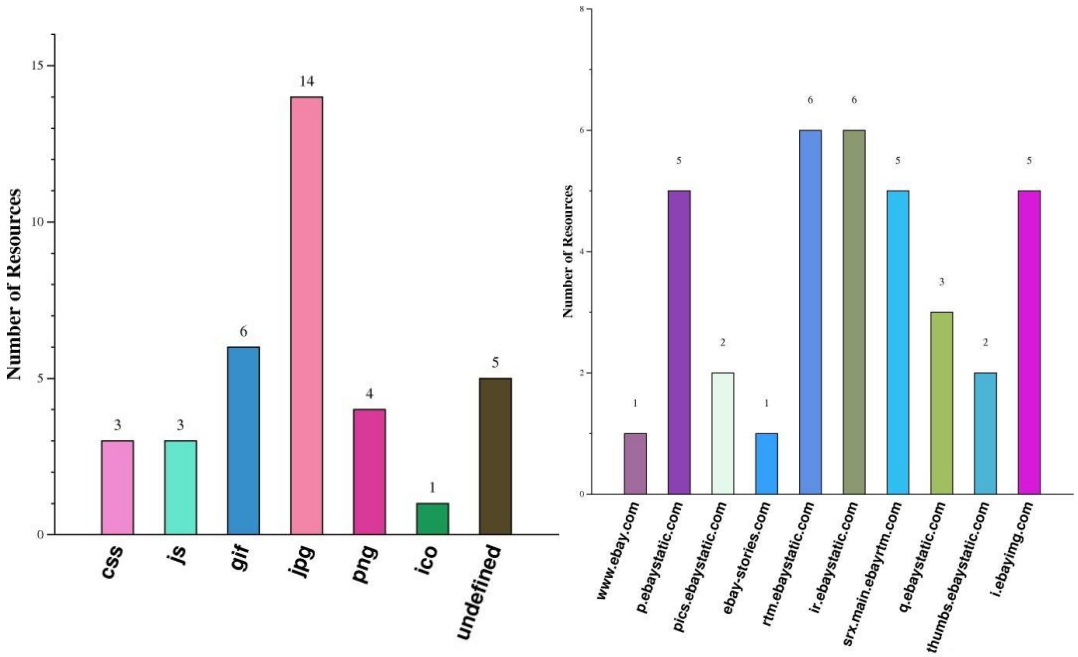
ebay.com
msn.com
ifeng.com
microsoft.com
myspace.com

Table 2: Websites to be presented

The websites that are presented below are shown in Table 2. The results of our analysis regarding these websites are presented in the Figures 6 - 10, below. More specifically, part (a) of every Figure shows in the x-axis the types of the components that are used from the website and in the y-axis the number of each type of these components. Part (b) of every Figure shows in the x-axis the content providers the components originate from. The y-axis presents the number of components every provider offers to the website.

As it can be seen in part (a) of each Figure there are some components that are characterized as "undefined". As "undefined" we categorize the components that have an unfamiliar type which our system was not able to recognize.

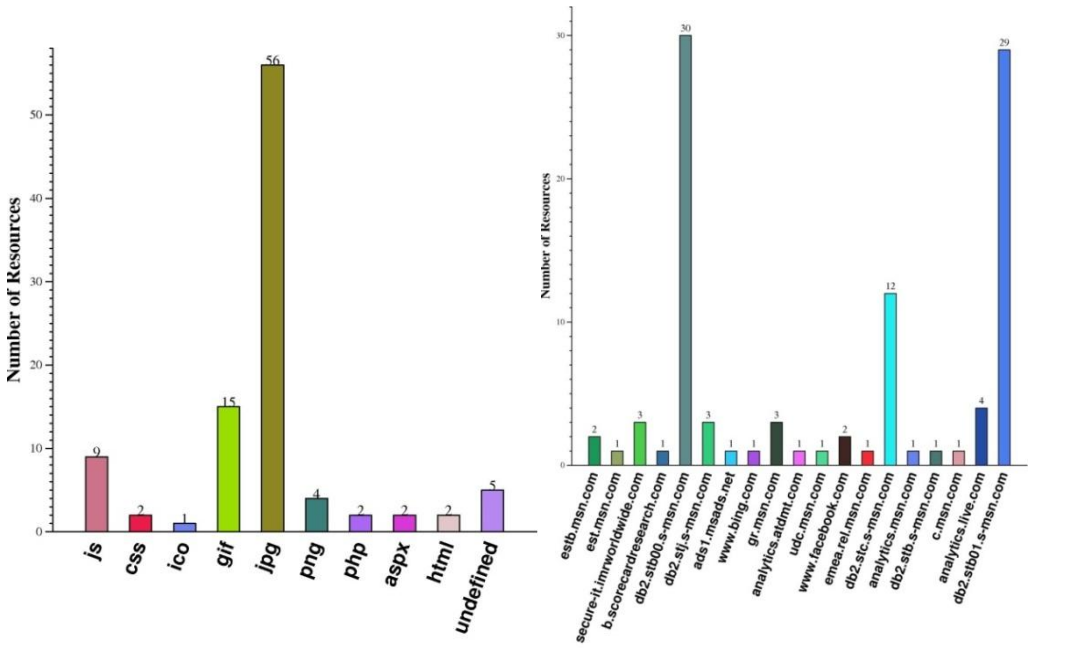
- *ebay.com*



(a)Type of contents *(b)Components' origin*

Figure 6*: ebay.com contents*

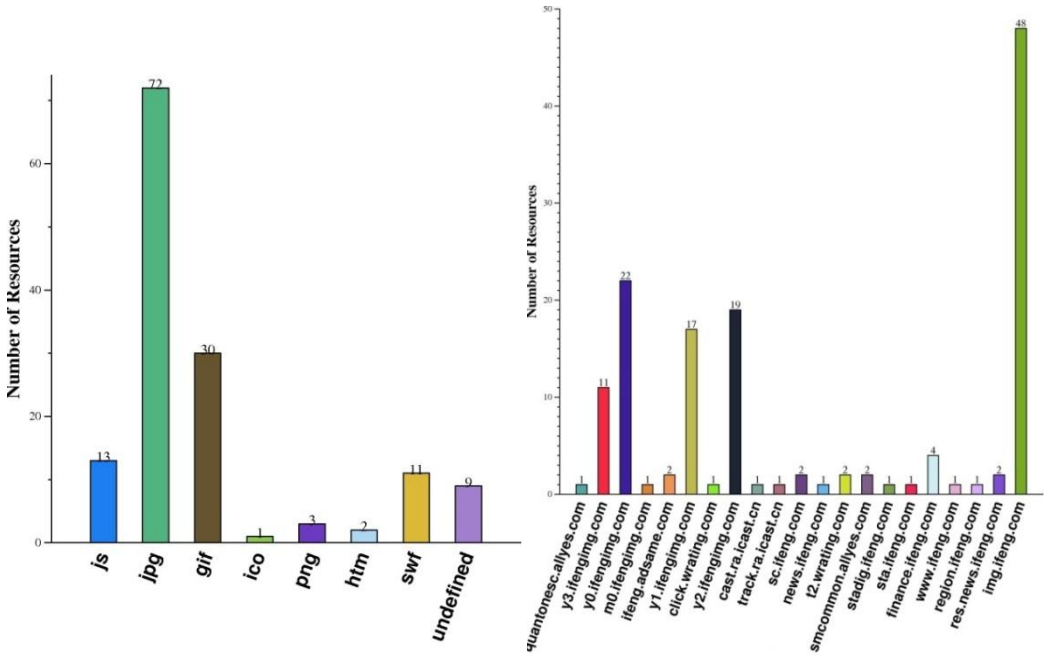
- *msn.com*



(a)Type of contents *(b)Components' origin*

Figure 7*: msn.com contents*

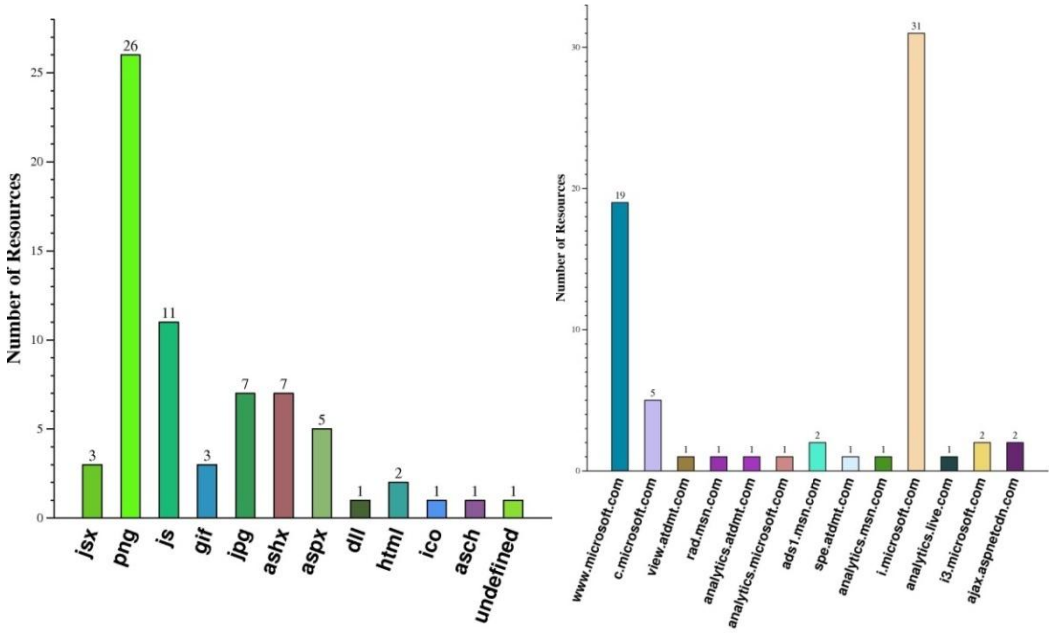
- *ifeng.com*



(a)Type of contents *(b)Components' origin*

Figure 8*: ifeng.com contents*

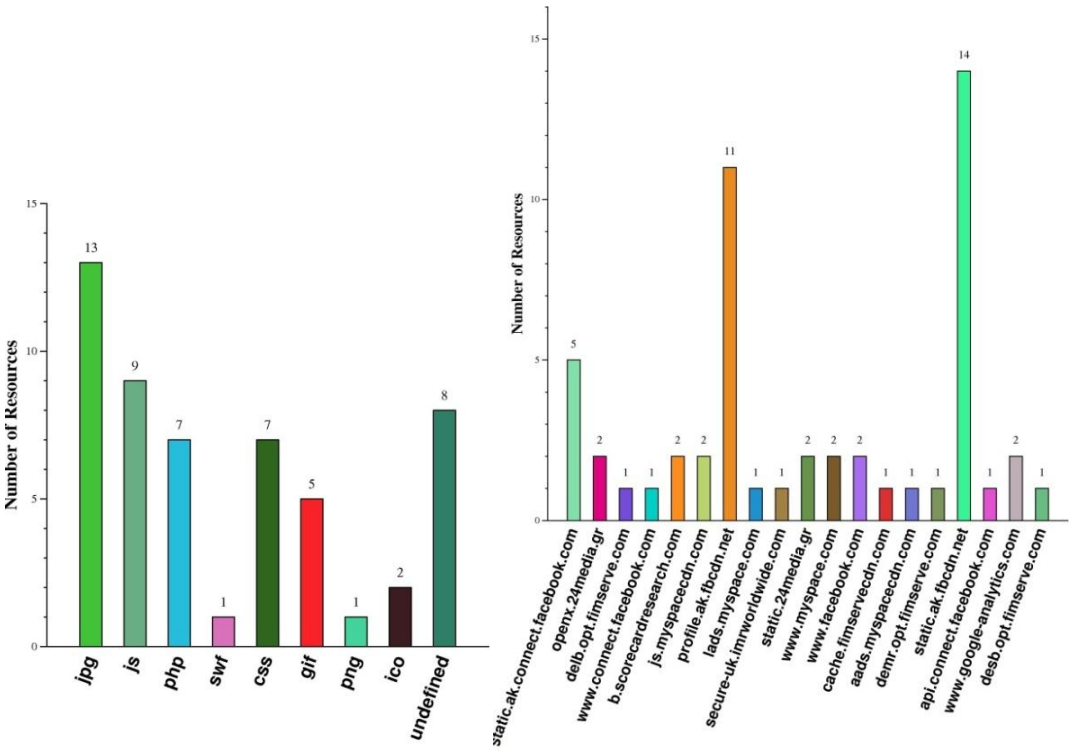
- *microsoft.com*



(a)Type of contents *(b)Components' origin*

Figure 9*: microsoft.com contents*

- *myspace.com*



(a)Type of contents *(b) Components' origin*

Figure 10*: myspace.com contents*

Figure 11 presents the results regarding the number of components every website uses from each class we have previously defined (inner, semi-outer and outer).

It is important to note that some websites in their graphs have zeros in all three classes. The possible reasons for these results might be that:

- The protocol that uses the specific website is HTTPS and not HTTP, so it is impossible to monitor packets exchanged with the server as in "paypal.com".
- The webpage was out of order when the experiment took place.
- The time (10 seconds) we specified for our experiment to run was not enough for the webpage to load from the browser we used.

Many websites which are very popular, since they appear as the 100 Top websites of Alexa, use an important number of components that originate from domains that do not belong to the owner of the parental website we have examined or to someone who is affiliated to him.

This phenomenon needs more attention because in many cases the content of these components is not checked from the parental webpage hence it is possible for a threat to be created. The threat could be either for the parental website or for some visitor.

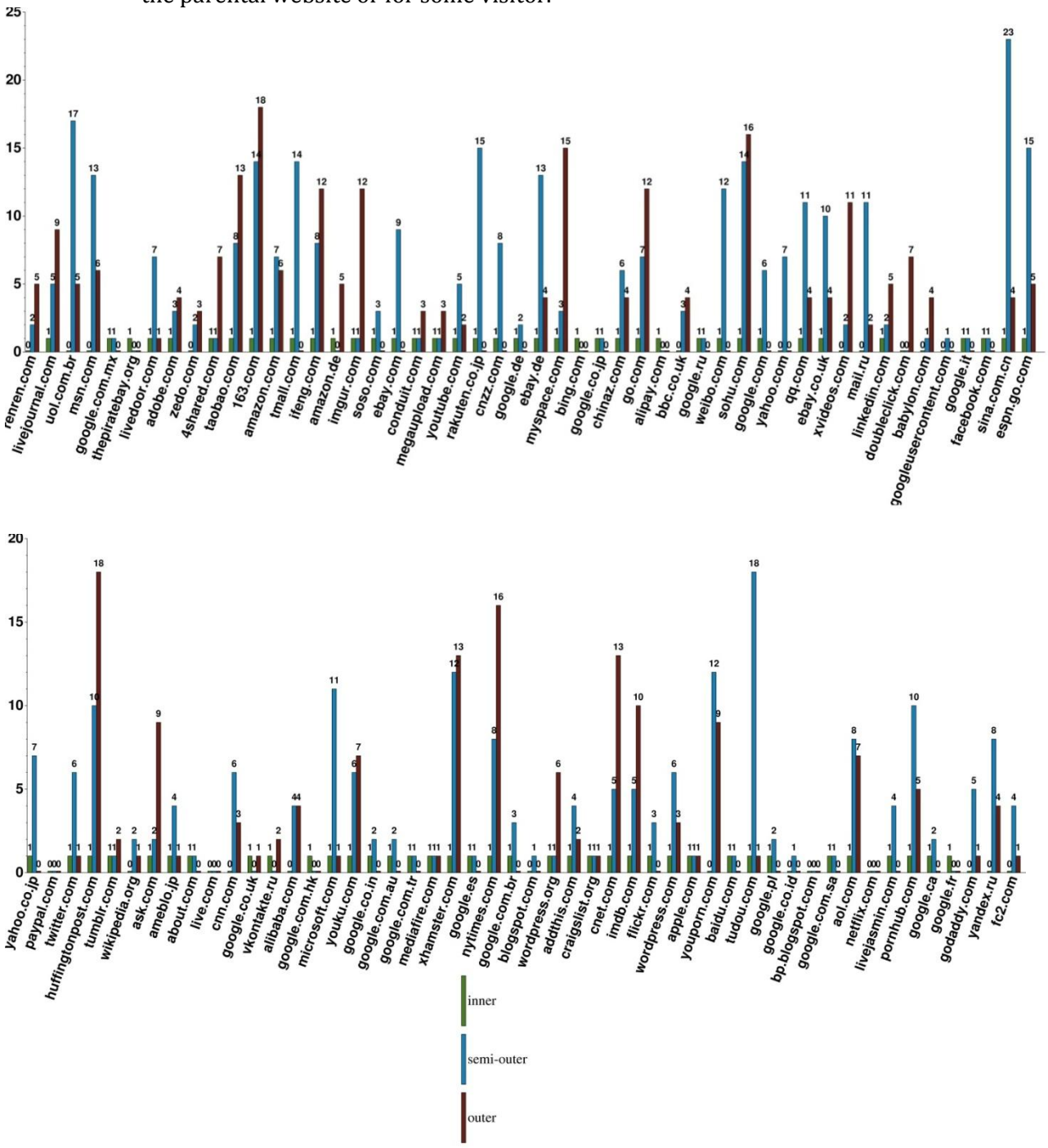


Figure 11*: Grouped Components' origin of Top 100 websites*

5.1. *Request Marking System*

In Request Marking System, the requests are marked before they are sent to the server. These requests are generated from events that are caused by the user's actions and they take place in the browser. This section will present a run from this system's execution. The scenario of our run is about requests that have occurred from an onClick event that was captured on our browser, after the insertion of a word in the search engine "google.com" from a user.

Below are shown two screen dumps which present the results of the proxy server for the requests that were sent to internet. A marked and a non-marked request header are presented. In the first dump a parameter after the capturing of the event has been added, while in the second dump there is not such a parameter, proving that it was sent without any user's action.

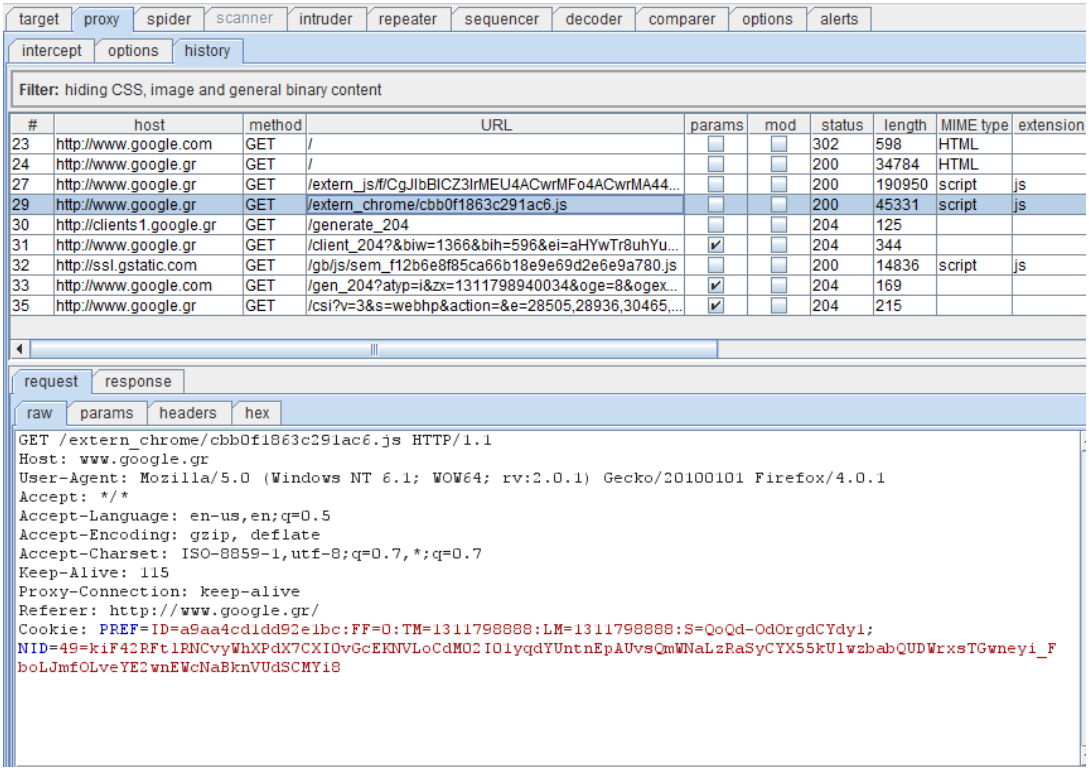


Figure 12*: Not marked request*

As it can be seen in Figure 11, the specific request did not pass through our system's event capturing section resulting in the absence of the marking parameter. This describes a request non-triggered from a user's action and declares that the information sent towards internet needs further investigation.

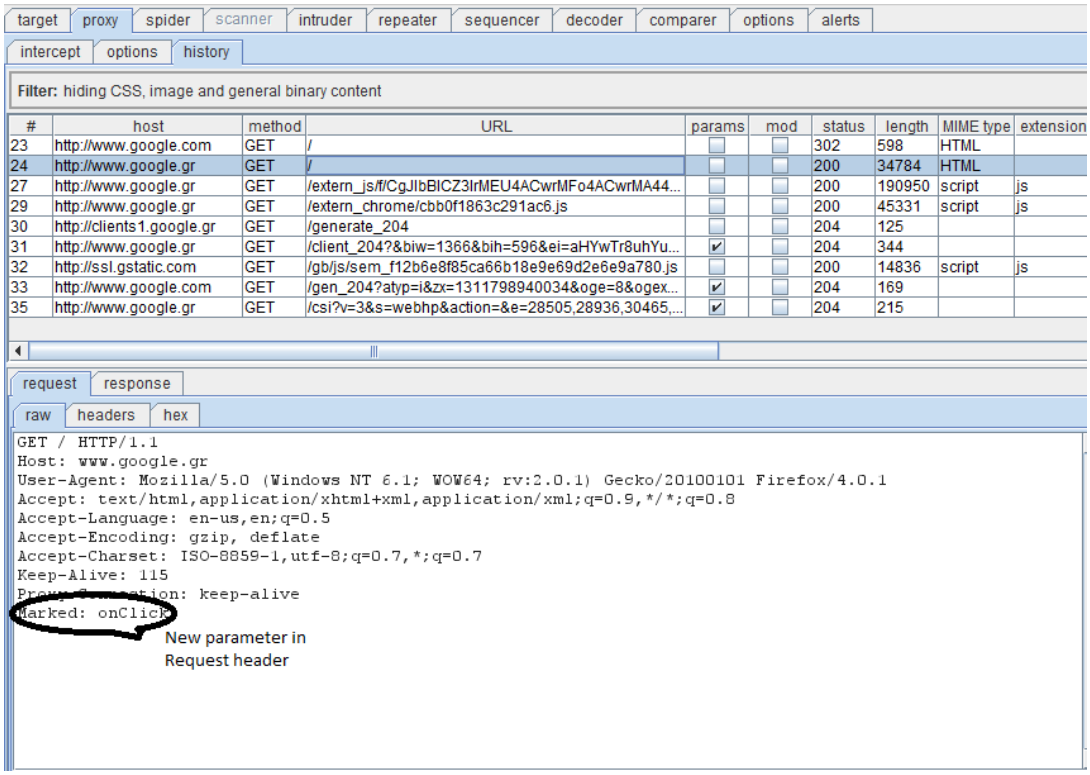


Figure 13*: Marked request*

In Figure 12 a request header to which the system has added a parameter can be seen. This parameter marks the request. In the variable's value field the type of the event that had happened in the user's browser and that had also triggered the request can be seen.

6. Conclusion and Future Work

In this report we have studied the problem of violating the user's privacy and we have witnessed not only the risen mistrust between a safe website and the components it uses from a third party but also a possible weak spot, which by exploiting can cause information leak from the user's browser. Moreover, we also have recorded possible attack scenarios created through exploitation of these components. In addition, we have presented an automated system that measures the content of the most popular sites on web. Finally, we have suggested an idea capable of helping us control the requests that a browser makes to a Web-server and we have created a system that brings this idea to life.

In the future, we could further investigate the conclusions regarding the vulnerability of every request that is able to pass through the proxy without being marked by the second system. We could also extend the marking system of the requests by automatically produce statistical measurements during the observation of the proxy's packet flow.

7. Acknowledgments

I would like to especially thank Elias Athanasopoulos for all his contribution and effort put in this work.

8. References

[1] O'Reilly, Tim - 2005
What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software.
http://mpra.ub.uni-muenchen.de/4578/1/MPRA_paper_4578.pdf

[2]Hewlett-Packard Development Company - White paper - 2007
SQL injection: are your web applications vulnerable?
https://download.hpsmartupdate.com/asclabs/sql_injection.pdf

[3]Amit Klein, Sanctum Security Group - 2002
Cross Site Scripting Explained
http://crypto.stanford.edu/cs155/papers/CSS.pdf

[4] Jesse Burns - 2005
Cross Site Reference Forgery: An introduction to a common web application weakness.
http://www.isecpartners.com/files/XSRF_Paper_0.pdf

[5] Moheeb Abu Rajab, Jay Zarfoss, Fabian Monrose, Andreas Terzis - 2006
A Multifaceted Approach to Understanding the Botnet Phenomenon
http://www.cs.jhu.edu/~terzis/imc114f-aburajab.pdf

[6] Brett McLaughlin - 2005
Mastering Ajax, Part 1: Introduction to Ajax
http://www.ibm.com/developerworks/web/library/wa-ajaxintro1/index.html

[7] Duane Merrill - 2006
Mashups: The new breed of Web app
http://www.ibm.com/developerworks/xml/library/x-mashups/index.html

[8] *Alexa Internet, Inc.*
An amazon.com company *that* provides information about websites including Top Sites, Internet Traffic Stats and Metrics, Related Links, Online Reviews Contact Information and many more.
http://www.alexa.com/

[9] *Whois.net*
Domain-Based Research Services

http://www.whois.net

[10] *Tshark.*
Network protocol analyzer
http://www.wireshark.org/docs/man-pages/tshark.html

[11] *Dpkt.*
Module for packet creation/parsing, with definitions for the basic TCP/IP protocols.
http://code.google.com/p/dpkt/

[12] *Jgraph*
A non-interactive filter for plotting two-dimensional scatter, line, and bar graphs in PostScript.
http://web.eecs.utk.edu/~plank/plank/jgraph/jgraph.html

[13] *Gecko 2.0*
Free and open source layout engine used in many applications developed by Mozilla Foundation and the Mozilla Corporation
http://en.wikipedia.org/wiki/Gecko_(layout_engine)

[14] *XUL*
Mozilla's XML-based language that lets you build feature-rich cross platform applications that can run connected or disconnected from the Internet.
https://developer.mozilla.org/En/XUL

[15] *Burpsuite.*
Integrated platform for performing security testing of web applications
http://portswigger.net/burp/

[16] World Wide Web Consortium
Document Object Model (DOM)
http://www.w3.org/DOM/.

[17] James Burke - 2006
Cross Domain Frame Communication with Fragment Identifiers
http://tagneto.blogspot.com/2006/06/cross-domain-frame-communication-with.html

[18] Frederik De Keukelaere, Sumeer Bhola, Michael Steiner, Suresh Chari, Sachiko Yoshihama – 2008
SMash: Secure Component Model for Cross-Domain Mashups on Unmodified Browsers
http://domino.research.ibm.com/comm/research_projects.nsf/pages/web_2.0_security.smash.html/$FILE/fp332-dekeukelaere.long.pdf

[19] Jesse Ruderman - © 2011 Mozilla Developer Network
Same origin policy for JavaScript
https://developer.mozilla.org/En/Same_origin_policy_for_JavaScript

[20] US-CERT, a government organization - 2006
Microsoft Internet Explorer HTML Document object cross-domain vulnerability
http://www.kb.cert.org/vuls/id/883108

[21] Douglas Crockford - 2006
A Proposed Solution to the Mashup Security Problem
http://www.json.org/module.html

[22] Ian Hickson (Editor) - 2007
HTML 5. Technical report
http://www.whatwg.org/specs/web-apps/current-work

[23] Google
Gadget-to-gadget communication.
http://www.google.com/apis/gadgets/pubsub.html

[24] Tamper data - Adam Judson, Mozilla extension
https://addons.mozilla.org/en-US/firefox/addon/tamper-data/

[25] Adam Barth, Collin Jackson, John C. Mitchell – USENIX Security 2008
Securing Frame Communication in Browsers
http://seclab.stanford.edu/websec/frames/post-message.pdf

[26] Adam Barth, Collin Jackson – 2008 Stanford Web Security Research
Protecting Browsers from Frame Hijacking Attacks
http://seclab.stanford.edu/websec/frames/navigation/