國立交通大學

資訊科學與工程研究所

碩 士 論 文

利用隔離使用者內容實現輕量級跨站偽造要求防禦機制

Light-Weight CSRF Protection by Labeling User-Created Contents

研 究 生：宋穎昌

指導教授：謝續平　教授

中 華 民 國　九 十 九 年 六 月

利用隔離使用者內容實現輕量級跨站偽造要求防禦機制

Light-Weight CSRF Protection by Labeling User-Created Contents

研 究 生：宋穎昌　　Student：Yin-Chang Song

指導教授：謝續平　　Advisor：Dr. Shiuhpyng Shieh

國 立 交 通 大 學

資 訊 科 學 與 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

June 2010

Hsinchu, Taiwan, Republic of China

中華民國九十九年六月

利用隔離使用者內容實現輕量級跨站偽造要求防禦機制

學生：宋穎昌　　　　　指導教授：謝續平 博士

國立交通大學

資訊科學與工程研究所

摘 要

根據 OWASP 2010 TOP 10 的資料，跨站偽造要求攻擊已經在危險程度中爬升到第五名。由於社交網站的出現，使得一般使用者與網頁伺服器之間有了新的互動方式，瀏覽器本身也支援 AJAX 語法來增強互動性，但是也同時增進偽造要求攻擊的能力。倘若攻擊者使用 AJAX 來發動偽造要求攻擊，現有的防禦機制將可輕易的被攻破。

本論文提出了一個輕量級的防禦機制來解決新型態的偽造要求攻擊。我們將針對攻擊的行為特性來擬定解決方案，且為了減輕伺服器端的負擔，捨棄了過往字串過濾或者是重新編寫 JavaScript 的方法，而是利用隔離使用者所提供之字串來達成的防禦機制。把網頁中的內容分成可信任及不可信任，當這些內容對網站發出要求(HTTP request)時，皆標上相對應的標籤，讓伺服器端可以根據這些標籤來判斷要求是否具有危險性。最後根據隔離出來的可疑要求，觀察其是否已達成伺服器所提供具有隱私資訊的服務來決定要求是否為偽造要求攻擊。網站的管理者可自定詳細的規則以達成更有效且精準的防禦機制。在論文的最後，我們將實作出這個概念並且測量其效能來證明此方法之可行性。

Light-Weight CSRF Protection by Labeling User-Created Contents

Student: Yin-Change Song   Advisor: Dr. Shiuhpyng Shieh

Department of Computer Science and Engineering
National Chiao-Tung University

ABSTRACT

Interactive website is the current trend of the Internet, but it has also created another opportunity for Cross-site request forgery (CSRF/CSRF). According to OWASP 2010 Top 10[1], CSRF/CSRF was listed as one of the most serious web vulnerability. Unfortunately, current protection approaches are not suitable when CSRF attack can use AJAX (Asynchronous JavaScript and XML) under the same website.

This paper presents a light-weight CSRF protection approach by introducing quarantine and inspecting suspicious scripts to the server-side. Instead of filtering and rewriting, our solution is based on labeling mechanism which helps web server to distinguish malicious requests from harmless requests. Once the administrator of the website indicates the critical services, the services that contain sensitive data or privacy information about users, labeling mechanism can prevent CSRF attack effectively without changing user-created contents (UCC). At the end of this paper, we implemented the proposed scheme and evaluated the performance of the implementation.

誌　　謝

這篇論文能夠順利完成，首先要感謝我的父母，是他們給予我這個機會就讀研究所，有機會讓我追求更專業的學識與技能，並且在我遇到困難的時候，適時給予鼓勵以及教導我如何做出抉擇。在研究室裡，很幸運的能遇到繼偉學長、Vic 學長、Mashi 學長、佳純學姐、子逸學長，讓我能在思考上以及待人處事上有了新的想法與見解，並且有機會能更著學長們四處參加比賽，累積各式各樣的經驗與歷練更是我這兩年來獲得最珍貴的寶藏。

對於同學們：Fallman，花生，阿布拉，ALAN，以及許許多多的學弟妹們，很感謝你們能夠對包容我白癡、低級又缺乏幽默感的腦袋，這段時間以來辛苦你們了，如果改天想要回味一下，歡迎隨時 Call In。

同時也要感謝最偉大的助理與工程師們：明華、順瑩、季榆、真如、慧雯、Michael、SKY、Pokei、晃晃…等等，讓我不僅僅能夠專心的研究上，也學到了許多人生上的態度、想法，與哲理！

感謝宥伶能夠在我需要的時候給予我勇氣以及前進的動力，尤其是妳那充滿激勵作用的言詞，更是我求學生涯不可或缺的糧食，也給轉聯的夥伴們：你們都是最棒的，大家要一起走下去阿！！！

最後也是最重要的，我要感謝謝續平老師能讓我有這個機會進入實驗室與大家相遇，也在老師的指導下能夠順利完成這篇論文，感激不在話下，臨表涕泣，不知所云。

Table of Content

List of Tables

List of Figures

Chapter 1

Introduction

Cross-Site Request Forgery (CSRF/XSRF) is listed among the top 10 web vulnerabilities in 2010[1]. Although CSRF attack is not as well-known as XSS attack, it might cause more serious impact in Web 2.0 [2][3]. Social network websites combine many services together and any members can make contribution to the website's contents. Once CSRF can be used in conjunction with XSS (Cross-Site Scripting), the whole website could be compromised in a short time.

1.1 User-Created Content

For social network website, the interactions between users and web server become more complicated. According to the concept of web 2.0, website allows users modify website's content, as called User-created contents (UCC). Users can create, adjust, or delete the content of the web pages by themselves. Therefore, browser cannot distinguish UCC from the content provided by administrator of website and trust in all data from web server. Sometimes, members hope that they can create dynamic contents, so JavaScript is required for creating dynamic HTML. JavaScript helps users enrich the effects of their web pages and improve the communication ability among users, but it also leads to unauthorized script injection attack. In order to ensure the safety of web pages, administrator has to filter user input and eliminate suspicious strings. As a result, filtering policies must not be too strict; otherwise members cannot fully build web pages on their willing.

Secondly, a social network website merges services together, including album, blog, guestbook, online shop, and Web ATM (Automatic Teller Machine) service. Because services are integrated into one website, attacker can invade all services only if one of services can be compromised [41] [43]. In the past, a successful attack scenario is always from a malicious server to another website. If there are many web services, attacker cannot breach all services at one time. But now, services are combined together. Consequently, attackers take this advantage to inject malicious scripts as UCC into victim's website and wait for members to browse. This action means that vulnerability is no longer "cross-site". Thus, the detection mechanism in the past is not suitable for social network websites.

It should be noted that JavaScript provided by UCC is not always malicious. Part of scripts just show some animation or provide a better experience on user interaction. Despite of these scripts are harmless, current solutions would block or modify them due to the potential threat. These protection approaches are in conflict with the concept of web 2.0.

1.2 Cross-Site Request Forgery

Cross-site request forgery (CSRF/XSRF) is one of web application attacks. It has caused serious damage to famous websites [2][3]. Because the defense approaches of XSS are more robust [15][26][31][34][39][37][43], stealing cookie becomes impossible. Instead of stealing cookie, attackers take the feature of browser cookie adding which called session riding. In cookie-based management, once user logins a server and session key is not expired, the browser embeds cookie into HTTP request header automatically. Attacker can spoof the payload of HTTP request and then browser transmits request to the server [29]. With session riding, server cannot determine whether the request is made by user or not. CSRF can do anything that user can do, for example, send email, post article, modify profile, transfer money from an account, or other services. Generally speaking, CSRF can imitate the user's identity to achieve services which is provided by server.

According to the browser-side language which is used by CSRF, the types of attack can be classified into two types.

With HTML, CSRF can only send one HTTP request to server and cannot obtain secret information in web pages. Due to the property of browser, HTML and JavaScript will be dismissed once a HTTP request is issued. But HTML is not restricted by Same Origin Policy (SOP). This property will cause that attacker can be launched from another website instead of website which victims is browsing. And this kind of attack can be triggered without JavaScript. Therefore, the defense of XSS cannot detect or block it. For example, attacker can create a form with prefilled parameters and lure users to click. The server cannot distinguish the request which is made by user or attacker.

This kind of attack can be easily solved. Take apart a service into many steps and each step needs browser to send a HTTP request. Attack will fail because HTML can generate only one HTTP request. Another approach is to use additional information. The additional information is generated dynamically and cannot be obtained in advance, thus attacker must send multiple requests.

With AJAX (Asynchronous JavaScript and XML), attackers can send multiple HTTP request, as called multi-stage CSRF attack. AJAX is a new client-side technique which combines HTML, CSS, DOM (Document Object Model), XML, JSON (JavaScript Object Notation), and JavaScript together. HTTP requests can be sent asynchronously in the background without being noticed. AJAX can also customize part of HTTP request headers and read the whole HTTP response. These capabilities make AJAX as a tiny browser which can obtain all secret information in the web page and HTTP response header. As long as the malicious scripts can be designed well, attackers can pretend to be a normal user and achieve anything they want in a website

[28][30][40][42]. Even more, once the malicious scripts can be stored in the server's database, CSRF can acquire self-propagation ability.

In the current solutions, JavaScript filtering or rewriting is the best way to defeat multi-stage CSRF attack. AJAX belongs to JavaScript, so JavaScript restriction is equivalent to limit ability of AJAX. However, filtering or rewriting costs high overhead while translating from original JavaScript to safer JavaScript. Another disadvantage is that functions must be provided by server. In conclusion, these approaches decrease the functionality of original JavaScript.

1.3 Contribution

Due to the prevalence of vulnerability of web pages, websites need an effective and flexible protection mechanism. This paper presents a labeling mechanism which is a server-side CSRF protection approach with light-weight access control mechanism. Based on the concept of user-centered design and inspired by [34][37][45], server will affix labels to each HTTP request invoked by user-created contents. Administrator can monitor HTTP requests at server-side and decide whether these requests are malicious or not. The most important this is that labels will not touch user-created contents.

We observe that existing websites are tired of filtering or rewriting user-created contents because of unpredictable or unexpected parsing behavior. Therefore, we propose a novel protection approach without sanitization. We ensures that UCC are isolated correctly and server-side can know exactly all HTPP requests from UCC. To be compatible with original JavaScript defined by ECMAScript [18], labeling mechanism allows JavaScript syntax with little limitation instead of filtering or rewriting. Since fine-grained protection policies, labeling mechanism can also accurately block CSRF attack until sensitive data are accessed. We also formalize the labeling mechanism by RBAC (Role-Based Access Control) [22] for integrity. Labeling mechanism can help both administrator and members maximize the utilization of resources from website.

Chapter 2

Background

2.1 Cross-Site Scripting

Cross-site scripting (XSS) is the most common vulnerability in the web application. It makes use of the flaw in the web pages and attackers can inject malicious scripts into web pages which served by trust web server. When surfing these pages, browser will automatically execute malicious scripts in the page without victims' permission. Since the script has the same privilege as the users, attackers can acquire sensitive data from victims. XSS vulnerability has been classified into two categories.

One is called "Reflected XSS." This vulnerability has "non-persistent" feature, because it depends on user to trigger a series of attack. For example, the content of dynamic web pages depends on the user input. Attackers often inject the codes they want into the input field and lure the users to activate. Once users click on the link, or button, the malicious code executes immediately and reflects back to the users. The most common situation is that attackers send some messages which contain dangerous links or content to victims.

Another is called "Store XSS." Malicious script stores in the database and executes whenever users request data which is polluted. This kind of attack can be more significantly than reflected XSS because malicious script is rendered more than once and all visitors become victims while browsing the web page. And the most important thing is that attacker does not need to face to victims. For example, attackers post a message containing malicious scripts on the forum. As long as the malicious content has not been removed, every visitor will execute scripts when watching this page.

2.2 Same Origin Policy

To prevent XSS vulnerability, the most common method is to filter user’s input. It is effective in the traditional web site. Netscape also introduced “**Same Origin Policy**” (SOP) to enhance the resistance of XSS. This policy restricts a script to can only access the attribute and method from the same site. It prevents any information from one origin to another, but it has not limitation in the same origin.

An origin is composed of application layer protocol, domain name, and TCP port. Same origin policy means that these three values must be the same; otherwise, browser-side programming language, such as JavaScript, cannot access the method or properties of the website.

2.3 HyperText Transfer Protocol

Hyper**T**ext **T**ransfer **P**rotocol (HTTP) is an application layer protocol between browser and web server. It is based on request-response standard. Once browser wants to transmit data to web server or obtain information from web server, a HTTP request is issued by browser and a HTTP response comes from web server.

In a HTTP request, there are usually two methods for transferring data in the modern web application. GET requests a specific resource on the web server. If GET is used for operations, it would cause side-effect, such as information leaking, cross-site request forgery. Attackers can make use of HTML sending HTTP request with GET. Because of HTML is not restricted by any policies, browser would send requests as attackers expect. The second method is POST. POST can submit data to update resource in the web server. In the HTML, HTTP request with POST is harder to forge than GET. It can be generated by HTML form tag only, but AJAX can simply generate a HTTP request with POST or GET.

2.4 Document Object Model (DOM)

Document Object Model (DOM) is an important structure of HTML, XHTML, and XML. As W3C speaking [4],

"*The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents.*"

In other words, all objects in the web page are DOM elements. For convenient, JavaScript can access DOM freely, including events, HTML tags, and CSS (Cascading Style Sheets), etc. Moreover, JavaScript can transform a static web page into a dynamic web page by modifying DOM immediately.

Chapter 3

Related Work

In recent years, there are many solutions to prevent CSRF attack [33][39]. CSRF is an attack that tricks victims into browsing a web page that contains malicious scripts which can forge HTTP requests. At the server's point of view, establishing a well-defined access control mechanism to a service is the most important thing. According to this concept, solutions can be simply classified into following categories:

3.1 No Script Policy

Website usually does not believe scripts which are composed by users. Therefore, UCC are considered as suspicious and administrators do not allow users to upload scripts. Users are only allowed to write some text on the web page. Whenever JavaScript is going to upload, the server eliminates or blocks them by sanitization method on strings [15]. On the basis of this approach, any client-side script language will be blocked, including HTML, JavaScript, and AJAX. Hence, it is good to defeat any client-side scripts attack.

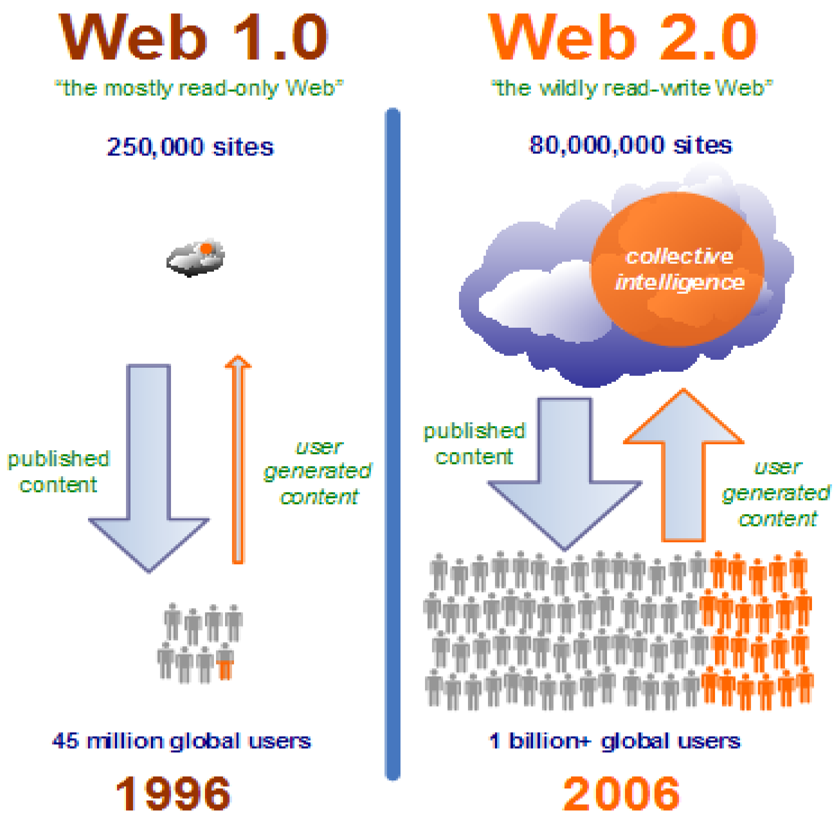


Figure 1, the difference of Web 1.0 and. Web 2.0 [25]

But in the web 2.0, the user interaction and experience is very important factor to a social network website. With too much limitation on the client-side script language, users cannot enrich their web pages; moreover the whole website is just like going back to web 1.0. Everything in the web page is static and lifeless.

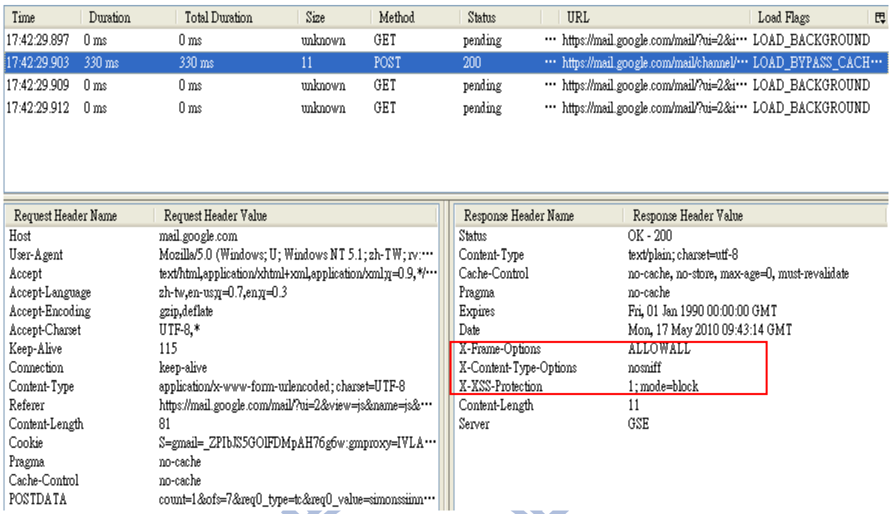
3.2 HTTP Header Modification

Based on the behavior of "cross-site", CSRF attack is launched from a malicious website to intrude another website. Hence, the solution is how to detect "cross-site" behavior. Without modifying client and server, Server can depend on HTTP Referrer header which stands for the previous URL browser surfed.

The potential problem is that HTTP Referrer header might leak the sensitive information that impinges on the privacy of users when cross website. An URL sometimes contains GET parameters, for example:

- http://www.google.com.tw/search?hl=zh-TW&q=secret

From the URL, we can know that user just enter http://www.google.com.tw/ and search a keyword "secret."



Time	Duration	Total Duration	Size	Method	Status		URL	Load Flags
17:42:29.897	0 ms	0 ms	unknown	GET	pending	···	https://mail.google.com/mail/?ui=2&i···	LOAD_BACKGROUND
17:42:29.903	330 ms	330 ms	11	POST	200	···	https://mail.google.com/mail/channel/···	LOAD_BYPASS_CACH···
17:42:29.909	0 ms	0 ms	unknown	GET	pending	···	https://mail.google.com/mail/?ui=2&i···	LOAD_BACKGROUND
17:42:29.912	0 ms	0 ms	unknown	GET	pending	···	https://mail.google.com/mail/?ui=2&i···	LOAD_BACKGROUND

Request Header Name	Request Header Value
Host	mail.google.com
User-Agent	Mozilla/5.0 (Windows; U; Windows NT 5.1; zh-TW; rv:···
Accept	text/html,application/xhtml+xml,application/xml;q=0.9,*/···
Accept-Language	zh-tw,en-us;q=0.7,en;q=0.3
Accept-Encoding	gzip,deflate
Accept-Charset	UTF-8,*
Keep-Alive	115
Connection	keep-alive
Content-Type	application/x-www-form-urlencoded; charset=UTF-8
Referer	https://mail.google.com/mail/?ui=2&view=js&name=js&···
Content-Length	81
Cookie	S=gmail=_ZPIbJS5GOlFDMpAH76g6w:gmproxy=IVLA···
Pragma	no-cache
Cache-Control	no-cache
POSTDATA	count=1&ofs=7&req0_type=tc&req0_value=simonssiinn···

Response Header Name	Response Header Value
Status	OK - 200
Content-Type	text/plain; charset=utf-8
Cache-Control	no-cache, no-store, max-age=0, must-revalidate
Pragma	no-cache
Expires	Fri, 01 Jan 1990 00:00:00 GMT
Date	Mon, 17 May 2010 09:43:14 GMT
X-Frame-Options	ALLOWALL
X-Content-Type-Options	nosniff
X-XSS-Protection	1; mode=block
Content-Length	11
Server	GSE

Figure 2, custom HTTP Header

Some papers propose custom HTTP header to figure out this problem [32][38]. For example, "Origin Header" prevents cross-site request forgery by modifying browser. This header only appears while sending a HTTP request with POST method. In the content of origin header, there does not contain sensitive data, like GET parameters or the path after domain name. Server can follow this header to verify that whether the source URL is dangerous or not.

This approach cannot apply to attack under the same website. Because the content of origin header has information about domain name, server cannot recognize who create the scripts and decide whether scripts should be block or execute. Furthermore, it is not convenient for users installing extra add-on or plug-in in order to browse a specific website.

3.3 Secret Validation Token

Instead of modifying HTTP request header, some researches implement a secret token mechanism to verify the legality of HTTP request, such as NoForge [5], CSRFx [6], and CSRFGuard [7]. When a client connects to server, this mechanism will generate and dispatch secret token to each other. That is both server and client have the same session key. Every time the client wants to communicate with server, secret token in the client-side will be verified by server. Secret token is useful for blocking forged HTTP request, but the location of secret token is a severity problem. In the client-side, browser embed secret token in the HTML tag, as following:

- <a>
- <form>
- <iframe>
- <button>
- <meta http-equiv="refresh" >

To modify these tags will cause some problems. First of all, it's hard to cooperate with DHTML. Dynamic HTML means the content of web page could be changed in the runtime. If a HTML form is generated at runtime, server cannot generate and send to browser immediately. As the result, the secret in client and server are asynchronous. Secondly, secret token will be embedded into "src" attribute which is equivalent to URL. But, URL is not protected well. Attacker can know the full URL by setting up a malicious website or using document.URL in JavaScript. Lastly, the most common problem is that all of these tags are DOM elements. In DHTML, JavaScript can access whole DOM tree. Thus, JavaScript will be prohibited when secret token exist in the same page.

3.4 JavaScript Restriction

As mentioned above, we can know that JavaScript is the biggest problem. Therefore, some researchers want to build a better JavaScript. These approaches depend on filtering or rewriting the dangerous functions or properties in the original JavaScript [36].

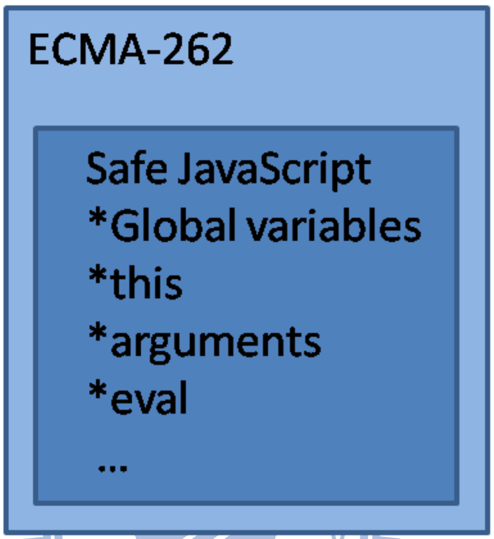


Figure 3, a safer subset of JavaScript

By doing this, define a much safer subset of original JavaScript. But there are some disadvantages in filtering or rewriting. For the user, functions are provided by server's API. Users cannot create new functions as they want. As a result, the freedom of JavaScript is restricted. To meet the security, although these papers always provide a formalization of safer-JavaScript, it can be broken sometimes. In [8], authors find vulnerabilities in FBJS [9] and ADSafe [10].

Rewriting is a hard work because of the properties of JavaScript. The most severe problem is the overhead of translation. In a client-server communication, time is the critical factor for users. However, filtering or rewriting is a time-consuming matter. Although its defense is very effective, users always hope for faster processing time.

Chapter 4

Proposed Scheme

To prevent CSRF attack, we establish a labeling function and construct UCC quarantine policy. The function is used to isolate user-created content and the policy can propagate labels with request from user-created content. CSRF attack is discovered when a request is labeled with untrusted and tries to access critical services which contain sensitive data or privacy information about users. In order to proof the integrity of proposed scheme, we use RBAC (Role-Based Access Control) to formalize and verify.

4.1 Main Idea

Browser always believes contents of the web page from web server even if authors of contents are not trusted by viewer. Once an attack can inject malicious scripts into web page, browser executes malicious scripts automatically without viewer's permission. This behavior causes severity problems, including CSRF attack. To prevent CSRF attack, we want to distinguish untrusted contents from trusted contents and prohibit untrusted contents from accessing web services which contain sensitive data, as show in Figure 4.

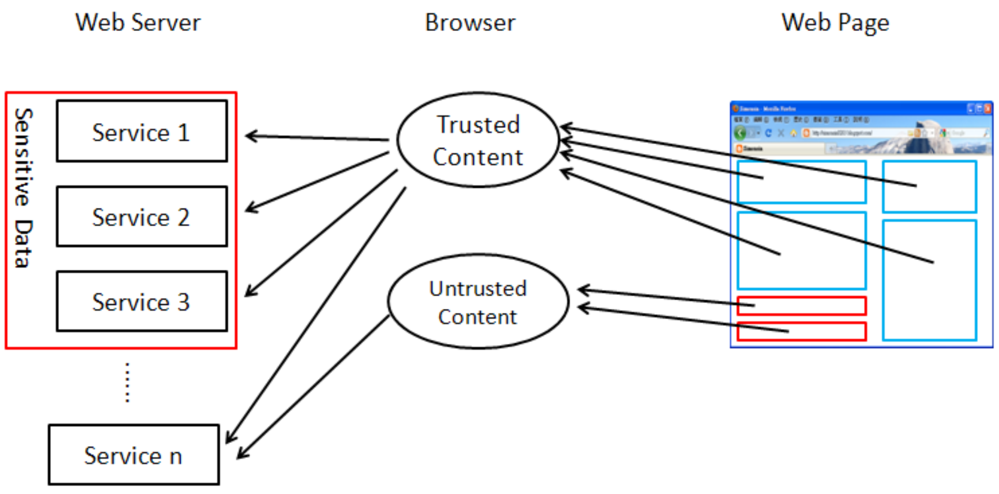


Figure 4, main idea

4.2 Observation

While surfing Internet, browser plays an important role between users and server. It has to manage session key which is called “cookie” and ensure the same origin policy is applied correctly. Server displays the content of web pages upon cookie. For example, if users want to see a web page with private information, they must provide relative session key which has enough rights. To avoid rights confusion, browser uses one cookie at a time in a website.

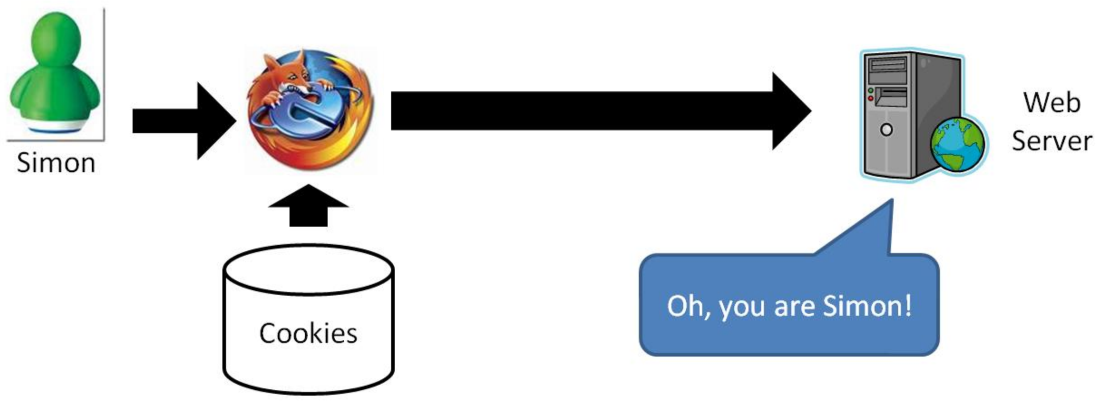


Figure 5, browser and cookie

Browser embeds cookie into HTTP requests automatically because of convenient, like Figure 5. Web server receives HTTP request and generates response which depends on cookie. In the process of communication, information which is acquired by web server comes from HTTP request instead of web page. Therefore, web server cannot exactly know the status of web page in the browser and attacker can just focus on how to build a flawless HTTP request rather than the whole web page.

Another observation focuses on the features of CSRF attack. The most important feature of CSRF attack is session riding. CSRF attack needs session to make a forge request with victim's identity. If the victim does not login website and web server does not know the identity of victim, CSRF attack is meaningless and cannot be successful.

4.3 Labeling

In a web page, contents can be classified into two types. One is created by administrator of web server, another is created by users. Contents created by users, as called "User-Created Content" (UCC), are the source problem of CSRF attack. But parts of UCC are harmless, such as contents which are created by viewer or does not contain malicious scripts. This kind of UCC should be classified into trust contents. The reason is that a CSRF attack is meaningless when both identity of attacker and victim are the same. Website can determine whether contents belong to current user or not by session, such as cookies. According to session, we can divide contents into two categories:

- *Trusted*: contents are created by administrator of website or current viewer
- *Untrusted*: contents are created by other users

Therefore, a labeling function is used to separate contents and ensure labels cannot be disrupted. The goal of labeling function is that every HTTP request must be labeled from browser, too. However, there may appear non-labeled HTTP request in some situation, such as user login, open new window, cross-site request, etc. These situations sometimes do not have session in HTTP request header, so web server should help them obtain cookies. But, if there exists session in HTTP request header, it can be considered as cross-site attack. In [11], the author proposed a simple and effective method for preventing cross-site attack. To build an essential page without any user's input can defeat cross-site attack before accessing normal web pages. We can extend this idea for blocking unknown HTTP request. Once a non-labeled HTTP request appears, server should redirect it to a non-UCC web page and take no parameters (GET or POST). By doing this, the page will not suffer CSRF attack from unpredictable user input and server can assign new cookie and label to browser. A cookie might contain following objects:

- User information: verify the identity of user
- Secret token: a temporary session key to valid freshness
- Session key: the common key used to keep login status

For different labels, cookies should not be the same.

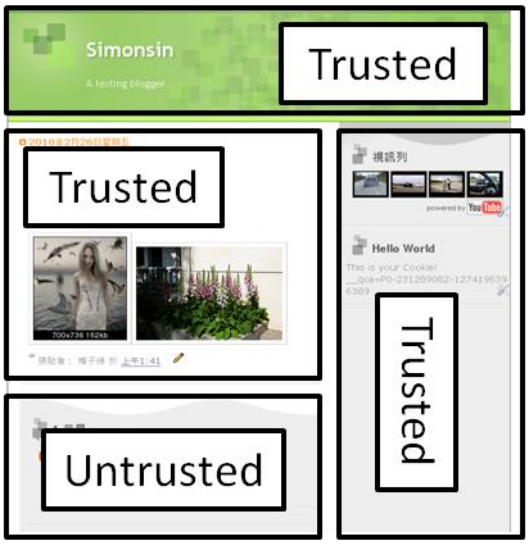


Figure 6, a web page with labels

In addition to labeling contents of web page, we also have to establish access control mechanism for labels. In a web page, there are many trusted and untrusted labels in Figure 6. If contents of trusted can be interfered by untrusted, attacker can inject malicious scripts into benign scripts. As a result, malicious scripts can issue HTTP requests with trusted label. In order to prevent this situation, labels have following restriction:

- Trusted label can access trusted and untrusted label freely.
- Untrusted label can only access contents with untrusted label.
- Once contents of trusted label are polluted by untrusted label, its label becomes untrusted, too.

Although labeling function can help server distinguish UCC from trusted contents, there are many challenges in implementation. Because of modifying server-side only, we must properly use JavaScript and built-in functionalities to reach restrictions and dispatch different secret tokens to each label. We will discuss these challenges in section 6 in detail.

4.4 Extending Labels

For multi-stage CSRF attack, labeling function is insufficient and needs more defense mechanisms. Because attacker and victim are under the same website, web server cannot depend on same origin policy or source information embedded into HTTP request. Because a multi-stage service requires a sequence of HTTP requests, as show in Figure 7, label should be the same during the process of accessing service. The process of accessing a web services is called transition path. Labeled transition path can help server determine that a series of requests are going to launch attack or just harmless.

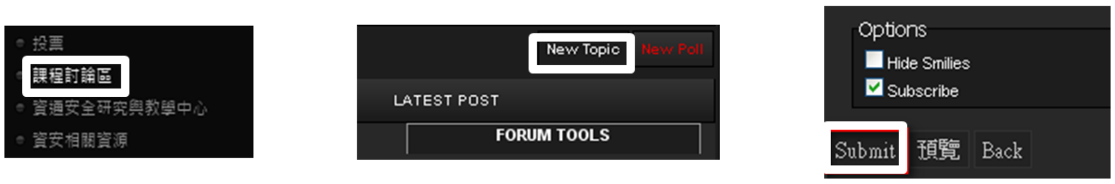


Figure 7, transition path of a service

HTTP defines lots of useful header and methods which can obtain information about browser status. Referrer header, GET and POST method in HTTP request can help web server know exactly the current URL and the next URL of browser.

- Referrer header: an URL stands for previous web page which issue this request
- GET method: retrieve information from URL
- POST method: upload resource to URL

If two URL are session-dependant, they should be considered as a minimum transition path. Session-dependant means that two pages share the same session. To build a complete transition path, web server connects these minimum transition paths upon sessions. By this way, we can catalog requests to build all transition paths, such as Figure 8.

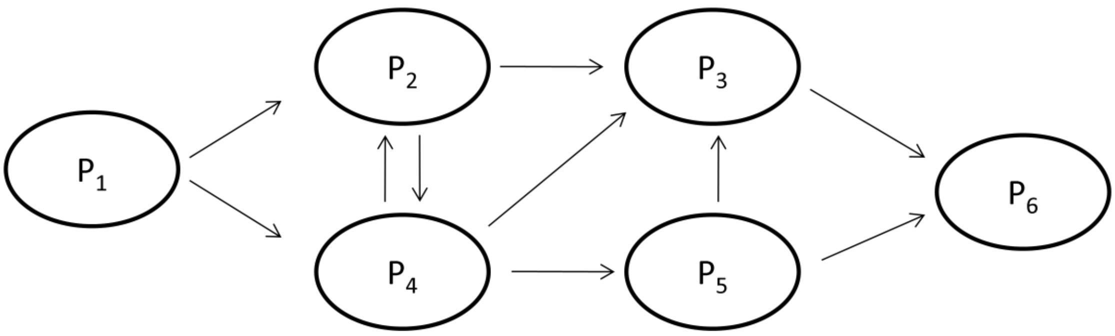


Figure 8, all transition paths of website

Once we have all transition paths, another challenge is to decide which transition path is multi-stage CSRF attack. By observation, attackers inject malicious scripts into UCC and wait for victim browsing. Therefore, we can know that a transition path invoked by a HTTP request with untrusted label should be noticed; nevertheless, parts of them are harmless.

To defeat CSRF attack accurately, administrator of website should define a subset of web services which will access sensitive data or privacy information about users. When an HTTP request is issued, web server should check the label of request every time to guarantee the integrity of critical services.

4.5 Determine CSRF Attack

As mentioned above, attacker injects malicious scripts into honest website and waits for victims to browse the vulnerability web page. Labeling mechanism can quarantine UCC and separate the untrusted part of contents. Once the malicious scripts launch attack, all requests are tagged with untrusted labeled. Furthermore, administrator creates the protection policies of critical services and stores them in database. Therefore, we can block CSRF attack effectively without modifying UCC if an HTTP request is tagged with untrusted label and wants to access critical services.

Chapter 5

Security Analysis

As mentioned above, our proposed scheme describes the concept of CSRF protection approach. To prove the integrity, we use role-based access control model (RBAC) to verify labeling mechanism. Although there are many good information flow access control model [12][13][27][35][43], our concept of preventing CSRF attack is simpler than system level access control. We focus on separating contents by trusted and untrusted authors instead of treating content as a unit. Hence we choose a role-based access control model.

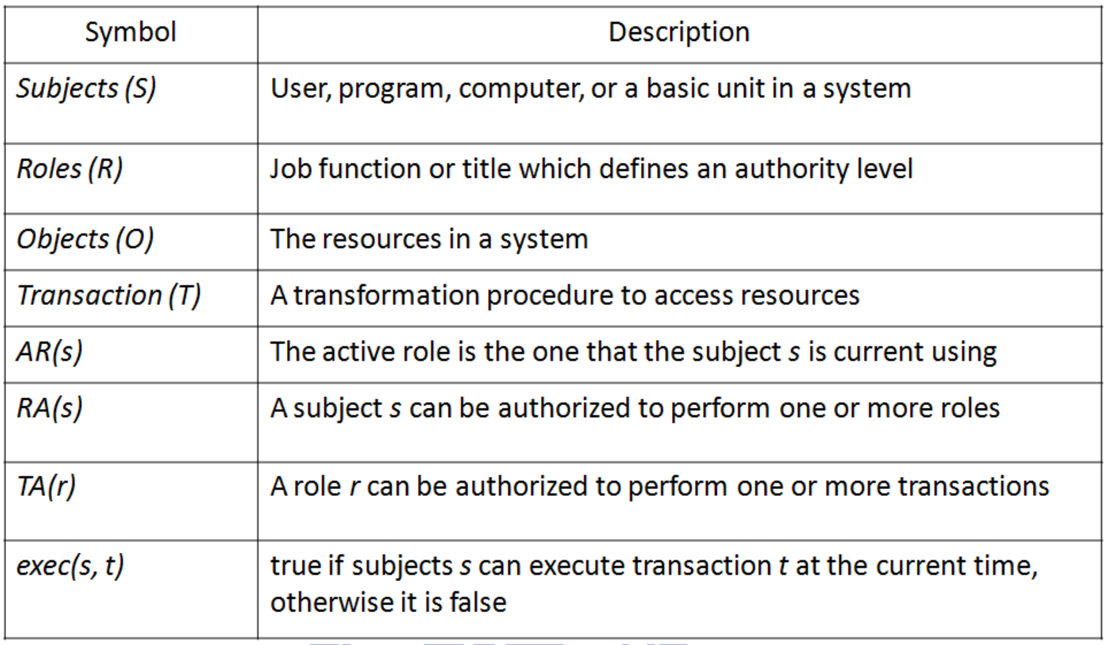
First of all, we introduce notations in RBAC and than these symbols are connected with labeling mechanism. We discuss the properties of security information, confidentiality, integrity, and availability in RBAC. In the end, we formalize CSRF attack and our proposed scheme by RBAC.

5.1 Notation of RBAC

RBAC (Role-Based Access Control) is a well-known access control model [31] and ensure the integrity and confidentiality of a system. In this section, we will introduce symbols of RBAC in Table 1.

There are three basic components in RBAC, Subject, Roles, and Objects. Subjects represent user, program or a basic unit, even a network computer. In our scheme, subjects are contents in a web page and contents can be classified by the identity of author. A role is a collection of job functions and dispatches permissions to each subject. We divide contents into two roles, trusted and untrusted, by authors' and current viewer's identity. The procedure of separating subjects is written as *AR(s)*. *RA(s)* means that one subject can perform one or more roles, but it can only use one role at a time.

Objects stand for the target of a program or the destination of execution path. Web services can be considered as objects because contents make use of browser for accessing resources in a website.



Symbol	Description
Subjects (S)	User, program, computer, or a basic unit in a system
Roles (R)	Job function or title which defines an authority level
Objects (O)	The resources in a system
Transaction (T)	A transformation procedure to access resources
AR(s)	The active role is the one that the subject *s* is current using
RA(s)	A subject *s* can be authorized to perform one or more roles
TA(r)	A role *r* can be authorized to perform one or more transactions
exec(s, t)	true if subjects *s* can execute transaction *t* at the current time, otherwise it is false

Table 1, description of symbols

Transaction is a transformation procedure plus a set of associated data items. To speak simply, it's a transition path of web pages for reaching an object. Each role may be authorized to perform one or more transactions, *TA(r)*. In addition, roles can be composed of roles. In other words, a higher level role can control all transactions of lower role. The most important symbol is "exec" which describes that whether a subject can execute a transaction or not under basic rules. Our proposed scheme can be drawn as Figure 9.

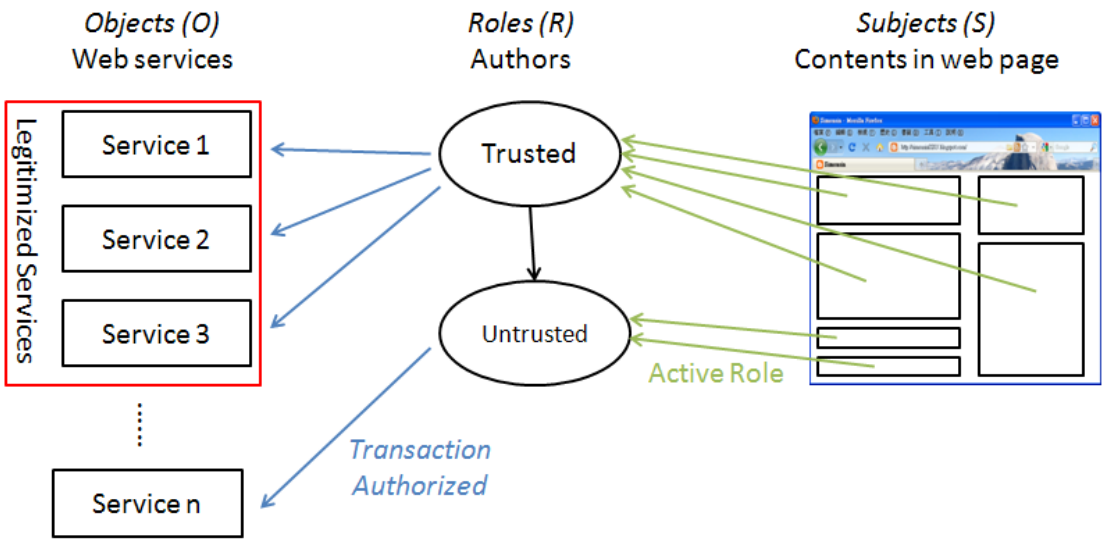


Figure 9, labeling mechanism with RBAC

5.2 Properties Analysis

In information security, there are three security goals: confidentiality, integrity, and availability. Now, we discuss the relationship between the three properties with role-based access control in our proposed scheme. As mentioned above, "exec" does not provide the rules of decision whether a procedure of accessing is legal or not. Therefore each transaction and subject is restricted with three basic rules:

- *Role assignment:*
  - $\forall s$: *subject*, t: *transaction*, $(exec(s,t) \Rightarrow AR(s) \neq \emptyset)$.

Role assignment rule complete the confidentiality property. Confidentiality means that the secret information should not be accessed by unauthorized individuals or systems. In RBAC, each subject must be assigned to a role or roles. That is to say, contents cannot access web services if browser does not have session with the website.

- *Role authorization:*
  - $\forall s: subject, (AR(s) \subseteq RA(s))$.

Role authorization rule can give the flexible ability to the system. In the reality environment, the active role of a subject changes frequently and this rule defines the changing is in scope. For labeling mechanism, the role of UCC depends on current viewer's and authors' identity. The changing of current viewer's identity affects the result of dispatching role for contents.

- *Transaction authorization:*
  - $\forall s: subject, t: transaction, (exec(s,t) \Rightarrow t \in TA(AR(s))$.

Transaction authorization rule ensure the integrity property. Integrity is the term used to prevent authorized roles from executing improper transactions. For example, CSRF attack disrupts the victim's session with website and mimics the victim's identity to access web services. In RBAC, transaction authorization rule ensures that all transaction must be executed with proper authorized role.

Availability assures that resources on the website should be available when they are needed. Administrator should give a guarantee of web services operating normally in any situation, including power outages, hardware failures, and system upgrades. However, this property is not in RBAC and CSRF attack does not break availability.

5.3 Integrity Property

In a CSRF attack, victim's browser will be forced to send requests to website by malicious scripts, as if requests were part of the victim's interaction with the website.

Browser leverages the session, such as cookies, and malicious scripts disrupt the integrity of the victim's session with website. To formalize CSRF attack by RBAC, the result shows as following:

- *CSRF := exec(s,t) is true only if t* $\notin$ *TA(AR(s)), where s*$\in$*subjects, t*$\in$ *transaction*

Obviously, CSRF attack violates transaction authorization rule. According to labeling mechanism, untrusted contents will be tagged with untrusted label and cannot access critical services. For current viewer, malicious scripts which can invoke CSRF attack are always embedded in untrusted contents. As long as we can ensure that malicious scripts cannot change or escape from the label, CSRF attack will never appear. Thus, the most important thing is to implement labeling mechanism with transaction authorization rule.

Chapter 6

Implementation

In section 4, we discuss that how to effective defeat CSRF attack with JavaScript functionality preserving at server-side. In order to prove that the proposed scheme is operable, this section will discuss the overview of labeling mechanism and focus on the challenges of implementation.

Labeling mechanism can be divided into two components in implementation, labeling function and quarantine policies. In Figure 10, we can see the position of components in reality situation. Note that, the labeling function and labels are generated from server-side.

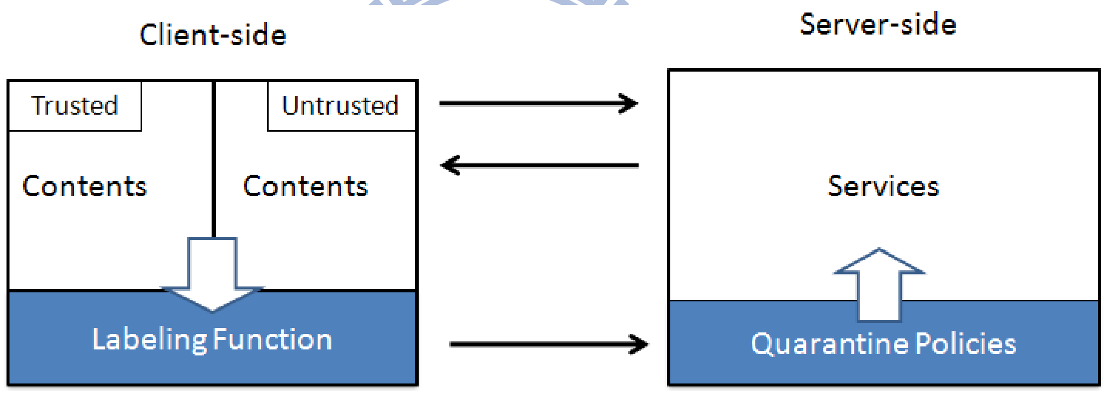


Figure 10, design overview of labeling mechanism

6.1 Labeling Function

Labeling function has three requirements. The first one is to distinguish untrusted contents from trusted contents. And the second is to ensure that this relationship will not be broken by untrusted contents. The third requirement is to enforce every HTTP request must be tagged with corresponded label.

About the first requirement, web server can preprocess the content of response. According to the discussion in section 4.3, if there are any data uploaded by other users, they should be tagged with untrusted label at first. In current website, web server stores data with author's identity, uploading time, and extra information in database. When a web page is requested by browser, web server can quarantine UCC by analyzing the identity of data. For untrusted part of UCC, web server should isolate untrusted contents by HTML iframe tag. In other words, untrusted contents are placed into iframes. The properties of iframe make sure that communication between trusted and untrusted label is still alive and all requests will be affixed with the same HTTP referrer header from iframe. Even if untrusted contents generate elements dynamically, HTTP referrer header will not be affected. Therefore, web server can dynamically generate response with iframe depend on identity provided by browser.

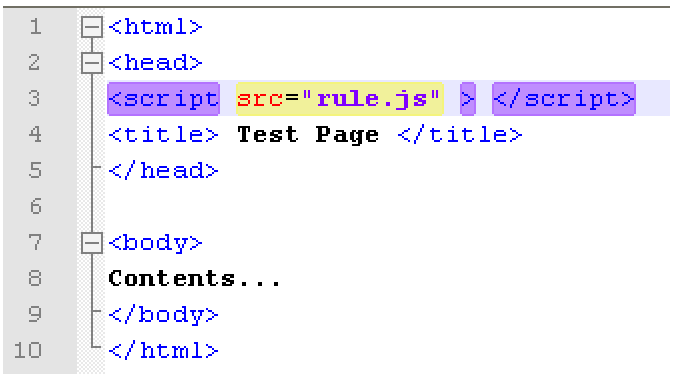
The second requirement is to restrict the access ability of UCC. Because iframe can communicate with parent node under the same website, we must setup a series of rules to restrict communication ability of UCC at client-side. Because our proposed scheme does not modify browser, we have to take use of JavaScript properties for restriction. JavaScript allows functions overriding and property redefine. For a configurable function or property, we can just assign a new object to override it. But some properties are non-configurable. In [14], authors use __defineGetter__, __defineSetter__, and Function.apply for changing the behavior of function and property. Redefining getter and setter is a useful skill for hiding non-configurable property, for example, "document.referer."

Functions	
XMLHttpRequest.open	Function override
XMLHttpRequest.send	Function override
Property	

document.cookie	Hide property
document.referrer	Hide property
window.parent	Assign new object
window.top	Assign new object
window.opener	Assign new object
window.self	Assign new object
document.parentNode	Return null by default

Table 2, modified functions and properties

In Table 2, there are four elements of window should be reconfigured because these elements let iframe access parent node's resource. As long as trusted label is parent of untrusted label, untrusted label has no chance to access trusted label. To make sure that UCC is restricted by our rules, we can create a rule.js which is placed at <head> section [14] [15] and put other UCC in <body> section. Although the javascript file might be loaded many times, browser has cache mechanism for reducing the overhead of loading the same file. With this method, there is no way to access trusted label in an iframe.



```
<html>
<head>
<script src="rule.js" > </script>
<title> Test Page </title>
</head>

<body>
Contents...
</body>
</html>
```

Figure 11, put rule.js in header section

But, here comes another problem – if untrusted label include a frame which contains trusted label, untrusted label can access trusted label. In reality situation, untrusted label can create another window object, an iframe object, and include trusted label page without any restriction. Because of no limitation on child node, untrusted label can access trusted label without obstruction. HTTP referrer header and function overriding can simply figure out this problem. To include a trusted label window means that browser has to send HTTP request and retrieve the sensitive data. Once a HTTP request is issued, browser will automatically embed current URL as HTTP referrer header and server can know whether request is trusted or not depend on referrer header. Current browser supports HTTP referrer header which cannot be modified by AJAX [16]. For privacy concern, we setup a rule for disable JavaScript access to document.referer. To avoid potential threat, web server should make use of HTTPonly [16] [17] for protecting cookie. HTTPonly disallow JavaScript access to a cookie.

Browser	Version	Prevents Reads	Prevents Writes
Internet Explorer	7 +	YES	YES
Internet Explorer	6(SP1)	YES	No
Mozilla Firefox	3.0.0.6+	YES	YES
Opera	9.50	YES	NO
Safari	4.0	YES	NO
Google Chrome	Beta	YES	NO

Table 3, browsers support HTTPonly

To meet the third requirement, we must know all methods that can send HTTP request to web server. With HTML, every tag which contains URL can be treated as a HTTP request, for example, <a>, <img>, <meta>, and <form>, etc. All of them will send HTTP request with referrer header. Hence, we can use referrer header to verify its label. By changing the path of URL, web server can easily know the label of request. Apache modules support “AliasMatch Directive” which can map URL-path to local files. With this module, administrator does not need really change all paths.

AJAX has customizing HTTP header ability, so we can override XMLHttpRequest function and attach a label on AJAX HTTP request. Overriding XMLHttpRequest function guarantees that the label cannot be overridden. Although referrer header can help us, customized header can ease off the overhead of server-side processing.

JavaScript defined “delete” operator can erase a function or property. When a built-in function or property is deleted, the function overriding loses efficacy and built-in function or property goes back. To prevent this situation, ECMAScript 5th edition [18] introduce “strict mode” [19] which can disable delete operator.

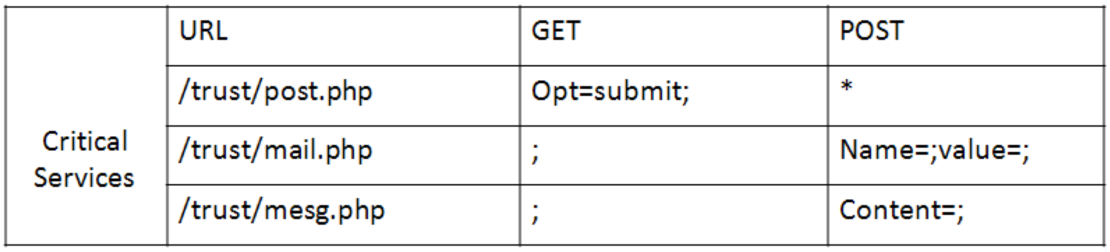
6.2 Identify Page Session

Now, all browsers have tab function. Each tab stands for a window and they do not interfere with each other. However, web server only recognizes HTTP request which does not contain information about tab. If we identify the session without considering tab, different tab requests cause false positive sometimes. In order to avoid this situation, cookie path can solve this problem. Different path can setup different cookie. This method is compatible with the 3rd requirement of labeling mechanism using the path of referrer header.

6.3 Check Policy

At server-side, administrator of website should define policies which describe critical services. In popular social network websites, they have self-checking mechanism between each web page. In other words, we only need to tag correct label on begin page and check end page. A policy format sketches as Figure 12.

The database of critical services is very convenient for administrator to add, delete, or adjust. Administrator only needs to setup the final web page of a service and the parameter of GET and POST. Once an untrusted label request is going to access one of web pages in critical database, the request should be block immediately.



	URL	GET	POST
Critical Services	/trust/post.php	Opt=submit;	*
	/trust/mail.php	;	Name=;value=;
	/trust/mesg.php	;	Content=;

Figure 12, critical policy format

When a CSRF attack is detected, the result of detection is forwarded to web server. Administrator can acquire the result by calling "begin_check()" function implemented in labeling mechanism. Therefore, this scheme can be deployed easily by adding three lines to each web page with PHP and construct untrusted iframe elements.

Chapter 7

Evaluation

We conducted experiments to evaluate performance overhead of our proposed scheme. Result of the evaluation fall into three categories: time overhead, memory usage overhead and defense effectiveness.

7.1 Environment of Experiment

First of all, we show the environment of evaluation. The server-side and client-side environment are described in Table 4.

Server-side:	
CPU	Intel(R) Xeon(R) 3050 @ 2.13GHz
Memory	1024 MB
Operating system	FreeBSD 8.0-RELEASE #1
Web server	Apache 2.2
	PHP 5.2.12 + MySQL 5.2
Client-side	
CPU	AMD Athlon 64 X2 4200 @ 2.2 GHz
Memory	2048 MB
Operating system	Windows XP SP3
Browser	Firefox 3.6.3

Table 4, environment of evaluation

Each test web page includes "check.php," an 8.8KB file without compression. And the untrusted iframe has to include a JavaScript policy file, rule.js. The size of rule.js is 4.4KB and the line of code is 146. It is also uncompressed.

7.2 Page Generation Overhead

To evaluation page generation overhead, we modify the web pages of Facebook [20] and MySpace [21] by ourselves. These social network website are very popular now. We create a service which contains four essential web pages. And browser surf the service over one thousand times.

The page generation overhead of modified Facebook service is range from 1.3% to 2.0%, but averaging a modest 1.6%. The modified MySpace service cost 1.8 % on average. The communication between browser and server is affected by the status of network connections, so we eliminate some irrational data which take less than 0.1% of result.

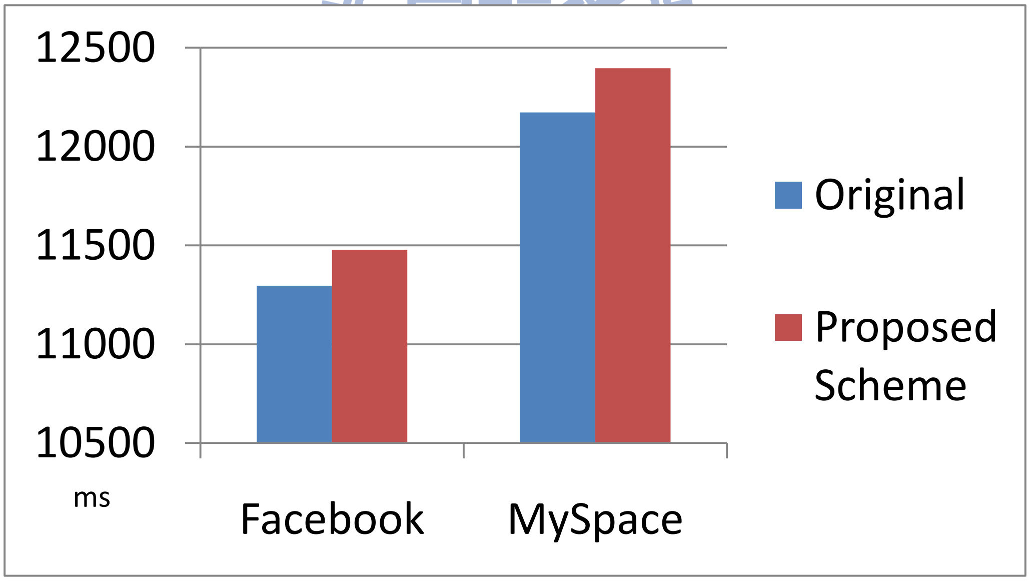
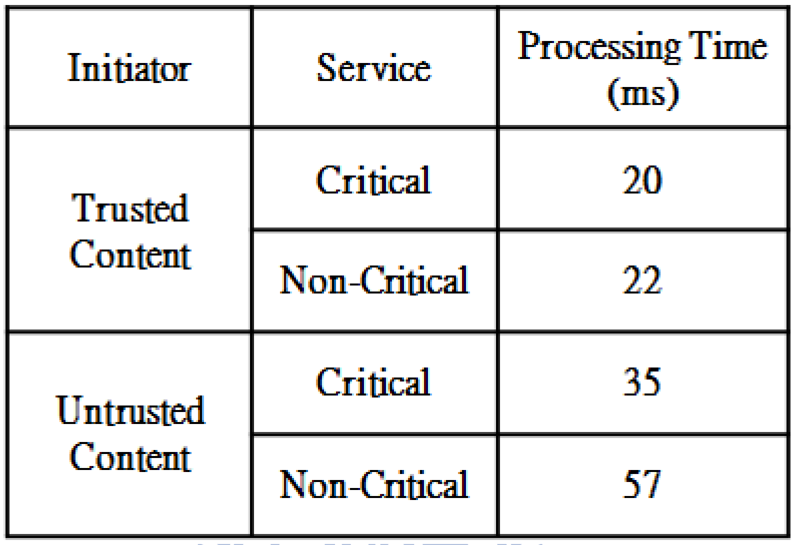


Figure 13, page generation overhead

We also measure the exact time of each access between labels and services. We can see that the execution time is very low and users will not feel the latency in Table 5. The execution time consists of enforcing JavaScript rule file, PHP execution at server-side and database query. But the last one, database query time, depends on the architecture and the size of the website. The database's size of popular social network website could be very huge and a data needs more time for querying, but database cluster system [23] and cloud computing technology [24] can decrease execution time effectively and make database query in real time.

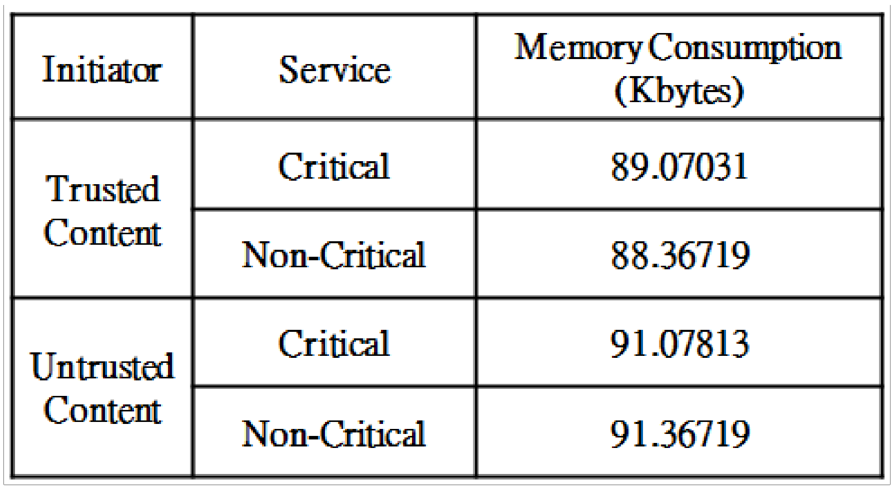


Initiator	Service	Processing Time (ms)
Trusted Content	Critical	20
	Non-Critical	22
Untrusted Content	Critical	35
	Non-Critical	57

Table 5, processing time of each access

7.3 Memory consumption

To calculate the memory usage, PHP provides memory_get_usage function for measuring the amount of memory allocated to PHP. We calculate the difference between memory usage of original architecture and policy-enforced in Table 6.



Initiator	Service	Memory Consumption (Kbytes)
Trusted Content	Critical	89.07031
	Non-Critical	88.36719
Untrusted Content	Critical	91.07813
	Non-Critical	91.36719

Table 6, memory consumption of accessing services

Another memory usage is HTML iframe tag. Because of our proposed scheme, iframe is used to quarantine untrusted labels and a web page might contain many iframe tags. Therefore, we calculate the memory consumption of an iframe in Figure 14. Each iframe costs 0.15MB on average. And the number of iframe inside a web page needs 10 at most by our observation in current websites.

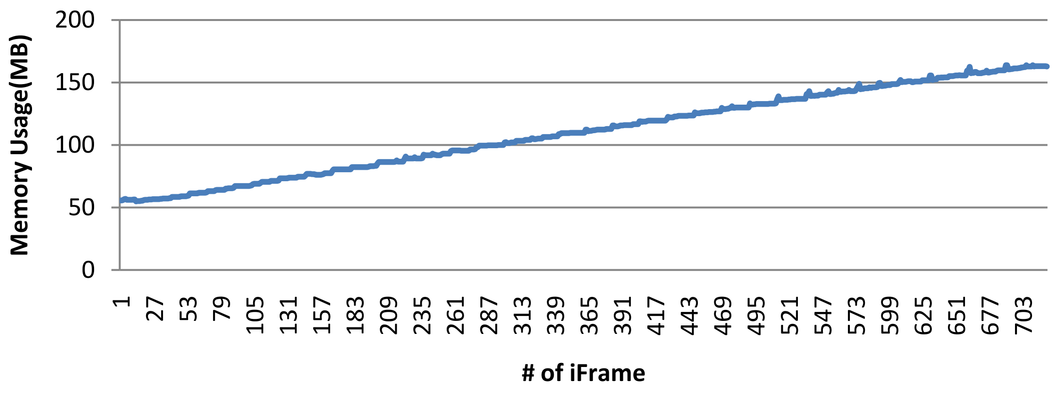


Figure 14, memory consumption of iframe

However, parts of memory consumption sometimes are affected by the content of web pages and the structure of website, especially in social network websites. In conclusion, memory consumption of both server-side and client-side are acceptable in current environment.

7.4 Defense effectiveness

The goal of our defense is to be effective against a variety of CSRF attacks. This section will discuss all situations about the preventing CSRF attack.

CSRF attack in different websites: A traditional CSRF attack means that victims must visit the malicious website first, the malicious script can forge an HTTP request and the browser of victim is forced to send this HTTP request. In our labeling mechanism, we can detect login CSRF by examining HTTP referrer header. In current browser, HTTP referrer header cannot be modified by JavaScript or HTML. Therefore, we can use HTTP referrer header to detect all forged HTTP requests from different websites, for example, [44] makes use of the flaw of Ebay. In this case, labeling mechanism can easily block forged requests by checking HTTP referrer header.

There is a similar CSRF attack called "Login CSRF attack [32]." Its concept is to override the cookie of the victims. By observation, login page is the most dangerous page of a website because it often lacks protection. However, HTTP referrer header is a built-in header, so the login page is protected as well.

CSRF attack in the same website: This kind of attack often cooperates with XSS and attacker does not need to set up a website. Attacker uploads malicious scripts to honest website's database at first. When victims surf the polluted web page, victim's browser will execute malicious scripts automatically. Attacker takes advantages of AJAX which can create HTTP request and customize HTTP request headers for forging HTTP requests. To prevent multi-stage CSRF attack, labeling function ensures that UCC is isolated and every HTTP request is tagged with corresponded label. Once all forged requests are captured, CSRF attack can be blocked easily.

Chapter 8

Conclusion

In this paper, we pointed out the root problem of CSRF attack and introduced the severity of CSRF attack in current social network websites. After surveying current solutions, we found a novel approach that could defeat multi-stage CSRF attack effectively.

According to the root of the problem about CSRF attack, we proposed a light-weight labeling mechanism protection approach. This approach takes advantage of built-in methods and properties to reduce performance overhead instead of filtering or rewriting the suspicious strings. The administrator only needs establish policies for critical services and inserts suspicious contents into iframe tag. To fully utilize benefits of web 2.0, we maximize the usability of JavaScript and AJAX with little restriction. Users can still use the original JavaScript and AJAX syntaxes and semantics under labeling mechanism. For convenient (in convenience), we provide a safety website without altering the browser. The members do not need to install any plug-in or add-on for a specific website. In conclusion, the proposed scheme could prevent CSRF attack without blocking website's interactive contents.

Chapter 9

Reference

[1] OWASP, "OWASP Top Ten Project," 2010. Available: http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

[2] Petko D. Petkov, "Google GMail E-mail Hijack Technique," 2007. Available: http://www.gnucitizen.org/blog/google-gmail-e-mail-hijack-technique/

[3] S. Kamkar, "I'm popular," 2005, description and technical explanation of the JS.Spacehero (a.k.a. "Samy") MySpace worm. Available: http://namb.la/popular

[4] World Wide Web Consortium, "Document object model (DOM) level 2 core specification," 2000. Available: http://www.w3.org/TR/DOM-Level-2-Core/

[5] N. Jovanovic, E. Kirda, and C. Kruegel, "Preventing cross site request forgery attacks," *Securecomm and Workshops, 2006*, pp. 1–10.

[6] Mario Heiderich. CSRFx, 2007. Available: http://php-ids.org/category/csrfx/.

[7] Eric Sheridan. OWASP CSRFGuard Project, 2008. Available: http://www.owasp.org/index.php/CSRF_Guard.

[8] S. Maffeis and A. Taly, "Language-based isolation of untrusted Javascript," *IEEE Computer Security Foundations Symposium*, 2009, pp. 77–91.

[9] Facebook, "Facebook JavaScript." Available: http://wiki.developers.facebook.com/index.php/FBJS

[10] D. Crockford, "ADsafe: Making JavaScript safe for advertising," 2008. Available: http://www.adsafe.org/

[11] F. Kerschbaum, "Simple cross-site attack prevention," *International ICST Conference on Security and Privacy in Communication Networks,* 2007, pp. 464-472.

[12] S.P. Shieh and V.D. Gligor, "On a pattern-oriented model for intrusion detection," *IEEE Transactions on Knowledge and Data Engineering*, vol. 9, 1997, pp. 661–667.

[13] S.P. Shieh, "A pattern-oriented intrusion-detection model and its applications," *Research in Security and Privacy*, 1991, pp. 327 -342.

[14] P.H. Phung, D. Sands, and A. Chudnov, "Lightweight self-protecting JavaScript," *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, 2009, pp. 47–60.

[15] V.N. Mike Ter Louw, "Blueprint: Robust prevention of cross-site scripting attacks for existing browsers," *IEEE Symposium on Security and Privacy*, 2009, pp. 331–346.

[16] World Wide Web Consortium, "XMLHttpRequest," 2009. Available:

http://www.w3.org/TR/XMLHttpRequest/

[17] OWASP, “HttpOnly - OWASP,” 2002. Available:
http://www.owasp.org.tw/index.php/HttpOnly

[18] Ecma International, “Fifth Edition of ECMA-262, ECMAScript,” 2009. Available: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf

[19] John Resig, “ECMAScript 5 Strict Mode, JSON, and More,” 2009. Available: http://ejohn.org/blog/ecmascript-5-strict-mode-json-and-more/

[20] Facebook, “Facebook.” Available: http://www.facebook.com/

[21] MySpace, “MySpace.” Available: http://www.myspace.com/

[22] D. Ferraiolo, DR. Kuhn, and R. Chandramouli, “Role-Based Access Controls,” *In Proceedings of the 15th Annual Conference on National Computer Security*, 1992, pp. 554-563.

[23] Oracle Corporation, “MySQL Cluster.” Available: http://www.mysql.com/products/database/cluster/

[24] The apache software foundation, “Apache Hadoop.” Available: http://hadoop.apache.org/

[25] Resnumerica, “Web 1.0 Vs Web 2.0,” 2006. Available: http://resnumerica.free.fr/nouveau-blog/?category/web1.0/

[26] P. Bisht and V. Venkatakrishnan, “XSS-GUARD: precise dynamic prevention of cross-site scripting attacks,” *Detection of Intrusions and Malware, and Vulnerability Assessment*, 2008, pp. 23–43.

[27] X. Lin, P. Zavarsky, R. Ruhl, and D. Lindskog, “Threat Modeling for CSRF Attacks,” *Proceedings of the 2009 International Conference on Computational Science and Engineering-Volume 03*, 2009, pp. 486–491.

[28] A.A. Al-Tameem, “The Impact of AJAX Vulnerability in Web 2.0 Applications,” *Journal of Information Assurance and Security*, 2008, pp. 240–244.

[29] D. Ahmad, “the Confused deputy and the domain hijacker,” *IEEE Security and Privacy*, 2008.

[30] H. Volos and H. Teonadi, “Study of security vulnerabilities in Web 2.0,” 2007.

[31] S Ravi, JC Edward, LF Hal, and EY Charles, “Role-based access control models,” *IEEE Computer*, 1996.

[32] A. Barth, C. Jackson, and J.C. Mitchell, “Robust defenses for cross-site request forgery,” *Proceedings of the 15th ACM conference on Computer and communications security*, 2008, pp. 75–88.

[33] M. Johns and J. Winter, “RequestRodeo: Client side protection against session riding,” *Proceedings of the OWASP Europe 2006 Conference, refereed papers track, Report CW448*, pp. 5–17.

[34] A. Yip, N. Narula, M. Krohn, and R. Morris, “Privacy-preserving browser-side scripting with bflow,” *Proceedings of the 4th ACM European conference on Computer systems*, 2009, pp. 233–246.

[35] J. Conallen, “Modeling Web application architectures with UML,” *Communications of the ACM*, vol. 42, 1999, p. 70.

[36] S. Maffeis, J. Mitchell, and A. Taly, “Isolating JavaScript with filters, rewriting, and wrappers,” *Computer Security–ESORICS 2009*, pp. 505–522.

[37] C. Karlof, U. Shankar, J.D. Tygar, and D. Wagner, “Dynamic pharming attacks and locked same-origin policies for web browsers,” *Proceedings of the 14th ACM conference on Computer and communications security*, 2007, p. 71.

[38] Z. Mao, N. Li, and I. Molloy, “Defeating Cross-Site Request Forgery Attacks with Browser-Enforced Authenticity Protection,” *Financial Cryptography and Data Security*, 2009, pp. 238–255.

[39] W. Zeller and E.W. Felten, *Cross-site request forgeries: Exploitation and prevention*, Citeseer, 2008.

[40] C. Jackson and A. Barth, “Beware of finer-grained origins,” *Web 2.0 Security and Privacy*, 2008.

[41] A. Barth, C. Jackson, and W. Li, “Attacks on JavaScript Mashup Communication,” *Proceedings of the Web*.

[42] B. Hoffman, “Ajax security,” 2006. Available: http://www.spidynamics.com/assets/documents/AJAXdangers.pdf

[43] J. Magazinius, A. Askarov, and A. Sabelfeld, “A Lattice-based Approach to Mashup Security,” *ASIAN ACM Symposium on Information, Computer and Communications Security*, 2010.

[44] Brian Prince, “eBay Security Vulnerabilities Found by Researcher.” Available at: http://www.eweek.com/c/a/Security/Researcher-Uncovers-eBay-Security-Vulnerabilities-684970/