Light-Weight CSRF Protection by Labeling User-Created Contents

Yin-Chang Sung, Michael Cheng Yi Cho, Chi-Wei Wang, Chia-Wei Hsu, Shiuhpyng Winston Shieh
Department of Computer Science
National Chiao Tung University
Hsinchu, Taiwan (R.O.C.)
{simonsin, michcho}@dsns.cs.nctu.edu.tw, cwwangabc@gmail.com, {hsucw, ssp}@cs.nctu.edu.tw

Abstract – **Cross-site request forgery (CSRF/XSRF) is a serious vulnerability in Web 2.0 environment. With CSRF, an adversary can spoof the payload of an HTTP request and entice the victim's browser to transmit an HTTP request to the web server. Consequently, the server cannot determine legitimacy of the HTTP request. This paper presents a light-weight CSRF prevention method by introducing a quarantine system to inspect suspicious scripts on the server-side. Instead of using script filtering and rewriting approach, this scheme is based on a new labeling mechanism (we called it *Content Box*) which enables the web server to distinguish the malicious requests from the harmless requests without the need to modify the user created contents (UCCs). Consequently, a malicious request can be blocked when it attempts to access critical web services that was defined by the web administrator. To demonstrate the effectiveness of the proposed scheme, the proposed scheme was implemented and the performance was evaluated.**

Keywords - cross-site request forgery, light-weight, Web 2.0, user-created contents

I. INTRODUCTION

Cross-Site Request Forgery (CSRF/XSRF) is listed among the top 10 web vulnerabilities [1]. Although cross-site scripting (XSS) came before CSRF/XSRF in the top 10 web vulnerability list, CSRF/XSRF is gaining attention in Web 2.0 environment [2][3]. In XSS, one of the severe cases is that the attacker could steal victim's session cookie to hijack victim's session and take over victim's account. To cope with the XSS cookie stealing problem, research work [15][25][33][36][44] has delivered promising results to counter against the cookie stealing problem. To overcome these defense techniques, cyber-criminals will have to switch to a different attack other than stealing user cookies, namely CSRF. In the cookie-based management, when a user logs in to a server and the session key has not expired, the client browser will embed user cookie into HTTP request header automatically. Instead of stealing user cookie, attackers can take advantage of this browser feature to achieve an attack called "Session Riding" or CSRF where an attacker can spoof the payload of an HTTP request and entice the victim's browser to transmit an HTTP request to the web server [28]. With session riding/CSRF, the server cannot determine legitimacy of the HTTP request. The forged HTTP request can cause a major problem since the request service may include important services, such as sending email, posting article, modifying profile, transferring money from the victim's account. That means CSRF attack can imitate user's identity to request for the services that are provided by web servers without stealing victim's cookie.

CSRF attack can be classified into two types. One is launched from a malicious website to an honest website. In this type, an attacker can only send an HTTP request to an honest website but no secret information can be obtained from the honest website. The Same Origin Policy (SOP) was design to prevent any cross-site HTTP request action that is caused by XSS. However, SOP is only effective against JavaScript, and HTTP request can be generated via HTML also. Therefore, a forged HTTP request generated by HTML can bypass the SOP. In an attack scenario, an adversary can pre-craft a HTML that sends an HTTP request to invoke a bank account transfer, and lure a rightful bank client to click on the link of the pre-crafted HTML. Thus the bank server cannot distinguish the rightfulness of the request, CSRF attack is achieved.

To cope with this type of CSRF attacks, a website administrator may break web services apart into several steps and each step requires the client browser to send an HTTP request. Using this method, the CSRF attack will fail since HTML can generate only one HTTP request at a time. An alternative approach is to use additional information, e.g. a secret token. The additional information is generated dynamically and it cannot be obtained in advance. Therefore, the pre-crafted HTML cannot send an HTTP request without knowing the additional information.

The other type of CSRF attack is based on JavaScript and AJAX (Asynchronous JavaScript and XML). It is called the "Multi-stage CSRF attack", which it involves a malicious script that generates multiple HTTP requests and secretly sends the generated HTTP requests asynchronously in the background. It can also customize parts of HTTP request header and read the HTTP request response. AJAX acts like a tiny browser which can it obtain all secret information from the HTTP request response headers and the payloads. With a well-designed malicious script, an attacker can pretend to be an authorized victim, accessing web services without restriction on the targeted website [27][29][39][41]. Furthermore, once the malicious script is stored in the database of a webserver, CSRF can also acquire self-propagation ability.

At present, JavaScript filtering or rewriting is the best solution to prevent the multi-stage CSRF attacks. AJAX is JavaScript based, and JavaScript restriction implies limiting the ability of AJAX. Filtering and rewriting JavaScript

978-0-7695-5021-3/13 $26.00 © 2013 IEEE
DOI 10.1109/SERE.2013.22

approach transforms the original JavaScript to a safer JavaScript with high overhead cost. This indicates that this approach reduces the functionality of the original JavaScript.

Web 2.0 is well adopted by social network websites, and social network website is one of the popular web services that are made available to the general public. For a social network website, the interaction between users and the web server adds complication to website design. One of the key features of a social network website is the UCC (user created content), in which users could contribute contents to the website. However, the identification of the user who provided the content can be vague or forged; and the client browser on the user-side cannot distinguish the UCC from the content that is provided by the administrator of the website. Other than identity of the UCC problem, user can also improve the content by adding JavaScript as part of the UCC to create DHTML (Dynamic HyperText Markup Language). JavaScript not only enables users to enrich the visual effects of their web contents, it also improves the communication ability between the users. Despite the benefits of JavaScript, it may also lead to an unauthorized script injection attack where CSRF is achieved. In order to ensure the safety of web pages, the web administrator has to filter user input and eliminate any suspicious strings. As a result, identifying the filter policies is a crucial task. Otherwise, members cannot contribute contents based on the advantages of Web 2.0.

A social network website often consolidates many web services together, services such as photo album, blog, guestbook, online shop, and money transaction service. Since these services can be integrated into one website, attackers can invade all services if one of the provided service is vulnerable [40][42]. Consequently, attackers take advantage of this to inject malicious scripts as part of a UCC into victim's website and wait for visitors to browse the malicious scripts. This malicious act means that the vulnerability is no longer a "cross-site" problem. Thus, the cross-site detection mechanism in the past is not a suitable solution for social network websites. It is also important to understand that JavaScript of a UCC is not always malicious. JavaScripts can be used to provide a better browsing experience for user interaction. Despite that these JavaScripts are harmless, current solutions may block or filter these JavaScripts due to the potential threat.

We observed that the existing websites do not adopt the idea of filtering or rewriting UCC since the parsing behavior can be unpredictable or unexpected. For this reason, we propose an effective and flexible protection scheme without sanitizing the JavaScripts. We ensure that the HTTP requests from a UCC are isolated and the web administrator can decide the access right of these HTTP requests generated by UCCs. To be compatible with the original JavaScript defined by ECMAScript [18], the proposed labeling mechanism, called Content Box*,* allows JavaScript syntax with small limitation instead of filtering or rewriting the JavaScript. Since fine-grained protection policies are enforced, the Content Box can also eliminate CSRF attack from accessing sensitive data/services. For integrity examination, we formalize the Content Box using RBAC (Role-Based Access Control) model [22].

The rest of the paper is organized as follows. Section II gives introduction to the related work. In section III, we propose the labeling mechanism for CSRF prevention and analyze the integrity of our scheme. The detail of security analysis is given in Section IV. Section V presents the implementation details. Evaluation and conclusion are given in sections VI and VII, respectively.

II. RELATED WORK

Various schemes have been proposed recently to prevent the CSRF attack [32][38]. The CSRF is an attack that tricks victims into browsing a web page that contains malicious scripts which can forge HTTP requests to pretend as the act of the victim. From the perspective of a web server, establishing a well-defined access control mechanism to a service is an important task to prevent CSRF attack. From the given information, the CSRF prevention scheme can be implemented on the server-side and/or the client-side. Based on the methodology of the schemes we categorized the related work into four categories.

A. No Script Policy

A website usually does not trust any scripts that are composed by the end-users. Therefore, any UCC contains scripts are considered as suspicious and web administrators do not allow end-users to upload any scripts. Posting plain text is the only permission that is allowed for the end-users. Whenever a script upload is detected by the web server, the web server eliminates or blocks the script by sanitizing the uploaded strings [15]. On the basis of this approach, any client-side script language will be blocked. These scripts may include HTML, JavaScript, and AJAX. Hence, it is an effective defense approach against client-side script attacks. In addition, there are web browser plugins available for disabling scripts of websites [45]. This is a client-side solution that enforces the client browser to comply with customized policies on website script execution.

The disadvantages of no script policy are that client-side script upload is totally disabled and/or the client browser on the client-side cannot view the effect of scripts. In Web 2.0, user interaction and experience are the key features to a social network website. With many restrictions on client-side script languages, users cannot enrich their web contents; moreover, the benefits of Web 2.0 features are not embraced.

B. HTTP Header Modification

Based on the behavior of "cross-site", the CSRF attack is launched from a malicious website to intrude a targeted website. Hence, the solution lies in how to detect the "cross-site" behavior. Without modifying both the client and the server, the server can rely on HTTP referrer header which it stands for the previous visited URL of client's browser. Hence, the cross-site HTTP request can be detected using the HTTP referrer header. However, the potential problem is that the HTTP referrer header may leak the sensitive information that impinges on the privacy of the end-users. For example, an URL sometimes contains GET parameters like the following,

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

From the provided URL, it reveals that the end-user had just searched keyword “secret” in google.com.tw.

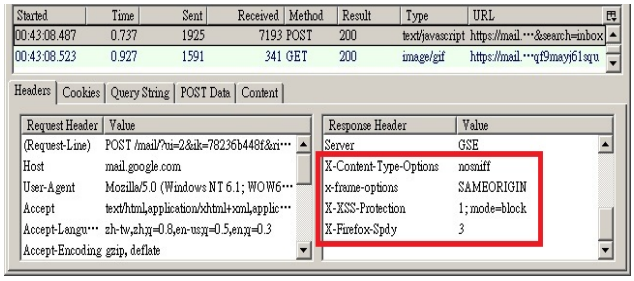

Fig. 1. Customized HTTP Header

Related work has proposed a customized HTTP header (see Fig. 1. for customized HTTP header example) to resolve the CSRF and privacy leakage problem [31][37]. “Origin Header” prevents the CSRF by modifying the client browser and the web server. This header will only appear when a client sends an HTTP request with POST method. In the content of the origin header, sensitive data is excluded for privacy preservation, e.g., the GET parameters, and the path after the domain name. A web server can use this header to identify the cross-site behavior. The disadvantage of this approach is that it failed to prevent the CSRF attack launched from the same website, since the cross-site behavior does not exist. Furthermore, it is not convenient for the end-users to install extra add-on or plug-in in order to browse a specific website.

C. *Secret Validation Token*

As an alternative of the HTTP request header modification methodology, some research works implement a secret token mechanism to verify the legality of an HTTP request [5][6][7]. When a client connects to a web server, the web server dispatches a secret token to the client browser dynamically. The secret token is transmitted from the client-side whenever the client is making HTTP requests to the web server. The transmitted secret token is verified by the web server to ensure the legitimacy of the HTTP request. Since the secret token is generated in real time, it is a useful scheme to defend against CSRF attack. However, the placement of the secret token is a severe problem. The placement of a client browser on the client-side embeds the secret token in one of the HTML tags as follows:

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

Some problem exists upon modification of these tags. First of all, it is difficult to cooperate with DHTML where the content of a web page can change at the runtime. If a HTML form is generated at the runtime, a web server cannot generate a secret token and send it to the client browser in real time. As a result, the secret token on the client-side and the server-side are asynchronous. Secondly, the secret token will be embedded into “src” attribute which is equivalent to an URL. Unfortunately, attacker can obtain the full URL by setting up a malicious website or using the document .URL property in JavaScript. Lastly, all of the mentioned HTML tags are part of DOM elements [4]. In DHTML, JavaScript can access all DOM elements in a web page. Thus, JavaScript must be disallowed whenever a secret token is used.

D. *JavaScript Restriction*

As aforementioned, a UCC that contains JavaScript is the root cause of the CSRF problem in Web 2.0. This leads to the research of building a safer JavaScript (See Fig. 2). This approach depends on filtering or rewriting the vulnerable functions or properties of the original JavaScript [35].

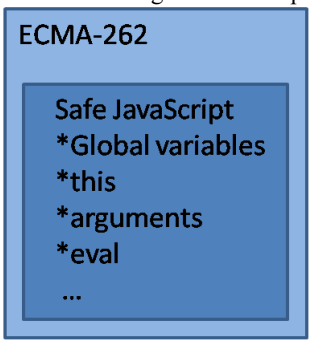

Fig. 2. A safer subset of JavaScript

Adopting this concept, the research defines a much safer subset of the original JavaScript. However, there are some disadvantages in the filtering or rewriting approach. First, the end-user can only use the functions that are provided by server’s API. Customized user function is prohibited using the filtering/rewriting approach. As a result, the freedom of JavaScript is restricted. Secondly, despite the fact that researches use formalism to prove the safeness of the safer subset of JavaScript, the safer subset of JavaScript is still vulnerable against attacks. Maffeis *et al.* [8] found vulnerabilities in research works [9][10].

Furthermore, rewriting is a difficult task based on the properties of JavaScript. The most severe problem is the overhead of JavaScript translation. In a client-server communication environment, response time is a critical factor. However, the filtering/rewriting approach is a time-consuming matter.

III. PROPOSED SCHEME

To prevent the CSRF attack, we propose a labeling mechanism called Content Box; the Content Box consists of a labeling function and UCC quarantine policies. The labeling function is used to isolate the UCCs, while the UCC quarantine policy enforces propagation rules for the labeled UCCs. The CSRF attack can be prevented using the Content Box when an untrusted UCC is trying to access a service that contains sensitive/private information. Furthermore, integrity of the Content Box is formalized and proved by using the RBAC model.

A. Main Idea

The contents of a web page can be divided into two different types. One is called the "trusted contents"; these contents are created by the web server administrator or the current viewer/user. Since these contents are created by the rightful owner, it is reasonable that the scripts within the contents are free from the CSRF attack. The other type is called the "untrusted contents" which are created by other users. Since these contents are provided by users other than the rightful owner, the scripts within these contents may cause the CSRF attack.

It is important to differentiate the contents of a web page since the client browser always trusts the contents of a web page provided by a web server even if the authors of the contents are not trusted by the client. In the Content Box, we intend to distinguish the untrusted contents from the trusted contents by labeling the contents and prohibiting the untrusted contents from accessing web services that contain sensitive data, as shown in Fig.3.

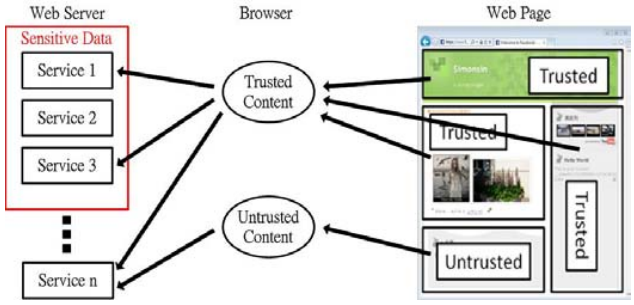

Fig. 3. An overview of the Content Box

When a client logs in to a website, a cookie is assigned to the current client browser. If the client wishes to make an HTTP request to the web site, the client browser will embed the assigned cookie into the HTTP request automatically, as shown in Fig.4. The content label relies on the embedded cookie to separate trusted and untrusted contents of the UCCs. In the present social network web server design, a social network web server records the author of messages, articles, or files. This feature can be used to identify the label of the UCCs.

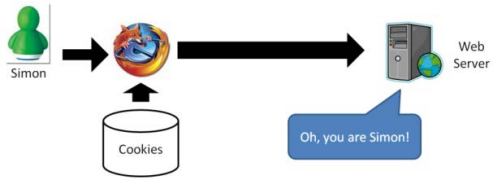

Fig. 4. The relation of the user cookies and the client browser

B. Labeling

In Web 2.0, the UCC is the source of the CSRF attack problem. However, most UCCs are harmless providing that if it is created by the current client. This kind of UCC should be classified into trusted contents since the CSRF attack rarely happens when both of the attacker and the victim are identical.

Labeling is used to differentiate the contents and ensure that every HTTP request is labeled, provide that the label cannot be disrupted by the client browser. However, a non-labeled HTTP request may appear under some conditions, e.g., user log in request, opening a new window, and cross-site request. For these non-labeled HTTP requests, the web server should assist these HTTP requests to obtain cookies for identification purpose. Inspired by Kerschbaum [11], we proposed a simple but effective method to block non-labeled HTTP requests. Whenever a non-labeled HTTP request is encountered, the server redirects the HTTP request to a non-UCC web page that takes no parameters (GET or POST). Therefore, the web page will not suffer the CSRF attack from unpredictable user input; the server can also assign new cookie and label to the client browser.

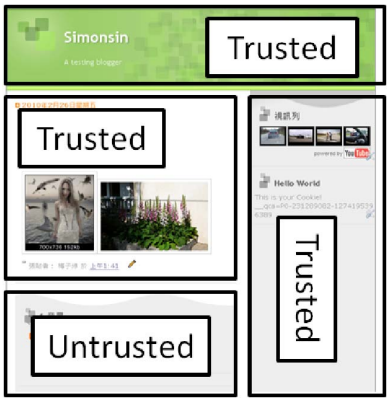

Fig. 5. A web page with labels

In addition to labeling the contents of a web page, an access control mechanism is required to patrol the accesses of web services. In a web page, it is common to have many trusted and untrusted labels, as shown in Fig.5. If the content of the trusted labels can be interfered by the untrusted labels, then the attacker can inject malicious script into benign content. As a result, a forged HTTP request with trusted label can be issued by an injected malicious script. In order to prevent this incident, labels have to comply with the following rules:

- Trusted label has the freedom to access the contents with trusted or untrusted label.
- Untrusted label can only access the contents with untrusted label.
- Once the contents with trusted label are contaminated by untrusted label, its label becomes untrusted.

The described access control scheme is simple and straight forward. However, many challenges exist during the implementation of access control scheme. With only server-side modification, we must use JavaScript and built-in functionalities to comply with the access control scheme. These implementation challenges will be discussed in section IV with details.

C. Extending Labels

For the multi-stage CSRF attacks, the labeling function is insufficient and hence stronger defense mechanisms are required. Since both the attacker and the victim have the same access right in the targeted website, the web server cannot rely on the SOP or source information embedded in the HTTP request to detect the multi-stage CSRF attacks. By observation, a multi-stage service requires a sequence of HTTP requests, as shown in Fig.6. We refer to this process as "transition path". A labeled transition path can assist a web server to distinguish between the multi-stage CSRF attack and the benign HTTP requests.

Fig. 6. A multi-stage service requires 3 HTTP requests

A minimized transition path can be constructed if two HTTP requests share the same session. To build a complete transition path, a web server connects these minimum transition paths based on shared sessions.

Once the transition path is built, the new challenge is to determine the multi-stage CSRF attack from the constructed transition paths. From our observation, attackers usually inject malicious script into untrusted contents and wait for a victim to browse the content. Therefore, the transition path triggered by an HTTP requests with untrusted label is considered as a multi-stage CSRF attack.

D. Determine CSRF Attack

To prevent the CSRF attack, the administrator of a website should define a set of services called the "critical services" which contain sensitive data and privacy information about users. When an HTTP request is received, the web server checks the label of the HTTP request to guarantee the integrity of the critical services. As described in section III, part B, labeling quarantines the UCCs and separates the untrusted UCCs. This methodology does not rely on filtering/modifying the UCCs and the CSRF attack can be blocked effectively provided that the label of an HTTP request is untrusted and the transition path involves a critical service.

IV. Security Analysis

As discussed in the previous section, the Content Box ensures the integrity of a website cannot be broken by the CSRF attack. To formally prove the integrity of the Content Box, we use the RBAC model to verify the integrity. Although there are many information flow access control models available [12][13][26][34][42], the Content Box is a simple model in comparison with system level access control models. The Content Box determines trusted content, untrusted content, and the services that can be accessed by untrusted contents. Based on the characteristics of the Content Box, we choose the RBAC model to formalize the Content Box.

A. Notation of RBAC

The RBAC model is a well-known access control model [30], and it ensures the integrity and the confidentiality of a system. In this section, we will introduce the symbols of the RBAC model in Table. 1.

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. Notation of RBAC

There are three basic components in the RBAC model: Subject, Roles, and Objects. A Subject represents a user, a program, a basic unit, or a network computer. In the Content Box, Subjects are the contents in a web page and the contents can be classified by the identity of the author. An Role is a collection of job functions and it grants permissions to each Subject. We divide the contents into two Roles, trusted and untrusted, by the identity of authors and current viewer. The procedure of separating the Subjects is written as AR(s). RA(s) means that a Subject can play one or more Roles, but it can only be one Role at a time. An Object stands for the target of a program or the destination of an execution path. Web services can be considered as Objects since the contents can access service resources on a website.

Transaction is a transformation procedure of a set of associated data items. Transaction path exists when a Subject reaches 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 Roles. The most important symbol is the "exec" which describes whether a Subject can execute a transaction under basic rules or not. The overview of the Content Box symbols can be drawn as Fig.7.

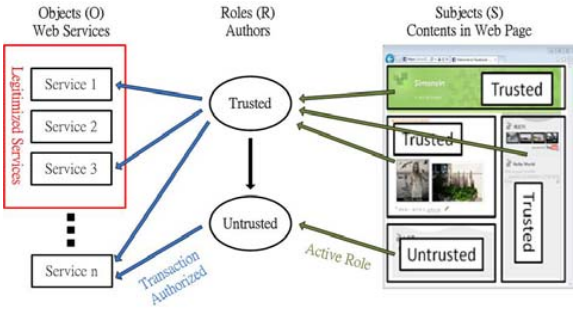

Fig. 7. Labeling mechanism using the RBAC model notation

B. Properties Analysis

In information security, there are three security goals: confidentiality, integrity, and availability. We will discuss the relationship between the three properties using RBAC model in the Content Box. Inspired by Ferraiolo *et al.*[22], three basic rules are formulated to restrict the transaction of an Subject:

1) Role assignment:

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

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

2) Role authorization:

$$\forall s: subject, (AR(s) \subseteq RA(s)).$$

Role authorization rule makes the system flexible. In a real world environment, the active Role of an Subject changes frequently and this rule guarantees the change of active Role is within the scope. For the Content Box, the Role of an UCC depends on the identity of the current viewer and the identity of authors. The change of current viewer's identity affects the result of dispatching Role for contents.

3) Transaction authorization:

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

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

Availability assures that the resources on a website should be available whenever needed. The web administrator should assure the correctness of web service operations under different circumstances, such as power outages, hardware failures, and system upgrades. However, this property is not described in the RBAC model and the CSRF attack does not corrupt availability. Therefore, we will not introduce the availability property for the Content Box.

In a CSRF attack, malicious script will force the client browser of a victim to send forge HTTP requests to the targeted website, as if the requests were part of the victim's interaction with the website. The client browser leverages the session, such as cookies, and malicious scripts disrupt the integrity of the victim's session with website. The CSRF attack can be formalized by the RBAC model as follows:

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

Evidently, the CSRF attack violates the transaction authorization rule. Referring back to the Content Box, the untrusted content will be tagged with untrusted label and such content has no access to the critical services. For the client, the malicious scripts that can invoke the CSRF attack are always embedded in untrusted contents. As long as we can ensure that a malicious script cannot escape from the regulation of the Content Box, the CSRF attack will be blocked.

V. IMPLEMENTATION

The Content Box implementation can be divided into two components, the labeling function and the quarantine policies. Fig.8 shows the placement of the components in a standard scenario. Note that the labeling function and the labels are generated from the server side.

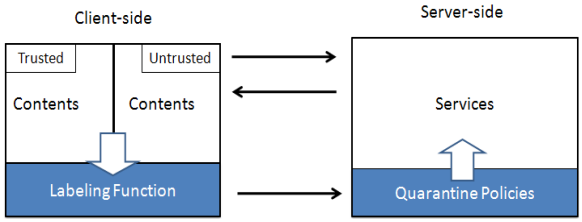

Fig. 8. Overview of the Content Box

A. Labeling Function

The labeling function has three functional requirements. The first requirement is to distinguish the untrusted contents from the trusted contents. In the present design of a website, a web server records the identity of authors, the upload time, and other information in a database whenever a user uploads data. We can use these recorded information to classify the UCCs into 'trusted' and 'untrusted' labels whereas the 'trusted' contents are the UCC provided by the web administrator or the current viewer. The rest of the UCCs belong to 'untrusted' label. For the untrusted UCC, the web server isolates the untrusted contents by using HTML iframe tag. That is, untrusted contents are placed into iframe and the iframe is embedded into the original web page. The properties of HTML iframe tag ensure that the internal and external communications of iframe stays available and all requests will be affixed with the same HTTP referrer header from the internal iframe. Even if untrusted contents generate elements dynamically, the HTTP referrer header will not be affected.

The second requirement is to ensure that the relationship of the labels will not be broken by the untrusted contents. The iframe can communicate with the parent node under the same website without restrictions. Therefore, we must restrict the communication ability of the child untrusted contents. Without client browser modification, we make use of the properties of JavaScript and browser built-in function to enforce this restriction. Function overriding and property redefine are the two functions that are available in JavaScript. We use these features to reconfigure functions and properties. Table 2 contains the functions and the properties that require reconfiguration. To achieve functions and properties reconfiguration, we assign a new object to override it. Unfortunately, non-configurable properties still exist. Phung *et al.* [14] used __defineGetter__, __defineSetter__, and Function.apply to change the behavior of function and

property. Redefining the getter and the setter is a useful skill for hiding non-configurable property, e.g. “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

There are four elements related to window (see Table 2) that require reconfiguration. These elements allow iframe to access the resource of the parent node. If a trusted label is the parent of an untrusted label, then the untrusted label will have the potential of accessing the trusted label using these four elements. To ensure that the untrusted UCC is restricted by the Content Box, we create a JavaScript policy rule.js file which is placed at the <head> section [14][15] and place other UCC in the <body> section. Although the JavaScript policy rule file might be loaded many times, a client browser has the cache mechanism to reduce the overhead of loading the same file rapidly.

Despite the fact that the aforementioned method eliminates the access of a child untrusted label to a parent trusted label, this introduces another problem – untrusted label includes a frame which contains trusted label. In this situation, the untrusted label can access the trusted label which is a child node of untrusted label. As a matter of fact, untrusted labels can create another window object, an iframe tag, and include trusted label contents without any restriction. Making use of the HTTP referrer header and function overriding can simply resolve this problem. To include a trusted label window, a browser has to send HTTP request and retrieve data with trusted label. Once an HTTP request is issued, the browser will automatically embed current URL as HTTP referrer header and the web server can identify that the HTTP request belongs to an untrusted label by reading the HTTP referrer header. Popular web browsers, as shown in Table 3, support HTTP referrer header, but it cannot be modified by AJAX [16]. However, the HTTP referrer header contains user privacy data. To protect user privacy, we setup a rule for disabling JavaScript access to document.referer. To avoid other potential threats, a web server can make use of HTTP only [16][17] for cookie protection. HTTP only disallows JavaScript access to the 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 HTTP only

The third requirement is to enforce that every HTTP request is tagged with the corresponding label. We must identify every method that can send HTTP requests to a web server. With HTML, every tag that contains URL can be treated as an HTTP request, e.g., <a>, <img>, <meta>, and <form>. These are tags that will send HTTP requests with HTTP referrer header. Hence, we can use the HTTP referrer header to verify the corresponding label. By changing the path of the URL, a web server can effortlessly identify the label of the HTTP requests. Apache modules support “AliasMatch Directive” which can map URL-path to a local file. Using this module, the web administrator does not need to change the paths in the web server.

AJAX has the ability of HTTP request header customization. Thus, we can override XMLHttpRequest function and attach a label to AJAX HTTP request. Overriding XMLHttpRequest function guarantees that the label cannot be overridden. Although using the HTTP referrer header can achieve the same goal, customized header can ease off the overhead of server-side processing.

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

B. Quarantine Policies

The HTTP process flow of the web server is shown as Fig. 9. When an HTTP request is received by the web server, the embedded cookie can be used to identify the identity of the requester. However, most of the recent browsers provide tab functionality where each tab stands for a window and tabs do not interfere with each other. A web server only recognizes HTTP requests without information regarding to the tabs. If we identify the session without considering the browser tabs, requests coming from different tabs will increase the false positive rate. To avoid misjudgment, the path field in a cookie can be used to resolve this problem. Different paths can setup different cookies and we dispatch different paths for different tabs. Alias Match Directive can fulfill this requirement and this method is compatible with the 3rd requirement of our labeling mechanism using the path of the HTTP referrer header.

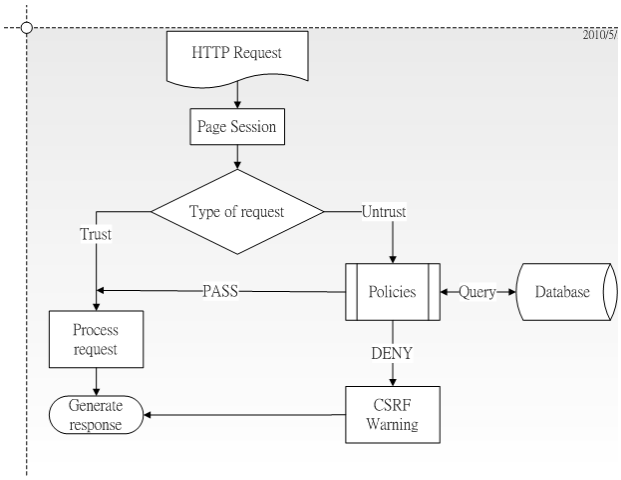

Fig.9. The HTTP request process flow of the web server

On the server-side, the web administrator should define policies which describe the final web pages of the critical services. If an HTTP request invoked by an untrusted content, the request will be affixed with untrusted label. Extending labels guarantee the correctness of label's propagation. A policy format sketches as Table 4.

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

Table 4. The format of critical policies

Once an untrusted label HTTP request is about to access one of the web pages in critical database, the HTTP request should be blocked immediately. When a CSRF attack is detected, the result of detection is forwarded to the web server. The web administrator can acquire the result by calling "begin_check()" function implemented in the labeling mechanism. Therefore, this scheme can be deployed easily by adding three lines to each web page with PHP and construct untrusted iframe elements.

VI. EVALUATION

We conducted experiments to evaluate the performance and the overhead of the Content Box. The results of the evaluations fall into three categories: time overhead, memory usage overhead and defense effectiveness. Table.5 shows the environment of the evaluations. Each tested web page includes a "check.php" 8.8KB file without compression. The untrusted iframe includes a JavaScript policy file, rule.js. The size of rule.js is 4.4KB, the line of code is 146 lines, and it is uncompressed.

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 5. The environment of the experiments

A. Page Generation Overhead

To evaluate page generation overhead, we use web pages of popular social network websites, Facebook [20] and MySpace [21], as reference samples and modify the sample web pages according to our proposed scheme. We create a service which contains four essential web pages, and make a client browser surf the service over one thousand times. The page generation overhead of the modified Facebook service ranges from 1.3% to 2.0%, but averaging a modest 1.6%. The modified MySpace service cost 1.8% on average (see Fig. 10). The communication between the client browser and the server is affected by the status of network connections. Therefore, we eliminate some irrational data which is less than 0.1% of result.

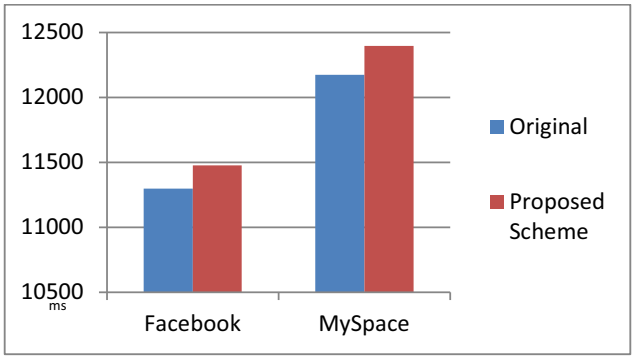

Fig.10. Page generation overhead

We also measure the latency of each access between the labels and the services. The execution time is very short and the client will not notice the latency as described in Table. 6. The execution time consists of enforcing JavaScript rule file, PHP execution on the server side, and database query. However, the database query time depends on the architecture and the size of the website. The database size of a popular social network website can be enormous and data querying could be time consuming. However, database cluster system [23] and cloud computing technology [24] can reduce the execution time effectively and perform 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.6. Processing time of each access situation

B. Memory Consumption

To calculate the memory usage, PHP provides memory_get_usage function for obtaining the amount of memory allocated to PHP. We use the provided function to calculate the memory consumption of the policy-enforced architecture in Table. 7.

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

Table.7. Memory consumption of accessing services

HTML iframe tag is another resource that consumes memory. In our proposed scheme, an iframe is used to quarantine the untrusted labels. Thus, many iframe tags may exist using our proposed scheme. Owning to this information, we calculate the memory consumption of an iframe in Fig. 11. Each iframe costs 0.15MB on average according to the calculation. Although the iframe number of current websites is closed to 10 at most according to our observation, we conduct our experiment based on many iframes to study the curve of memory consumption.

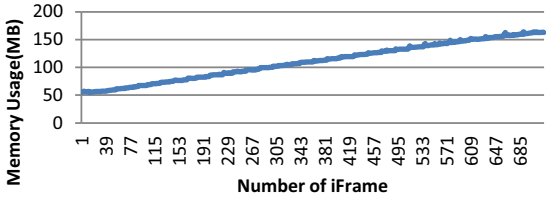

Fig.11. Memory consumption of iframe

Although we made our best effort to calculate the memory usage, part of memory consumption is also occasionally affected by the content of the web pages and the structure of the website. These attributes varies in social network websites. Taking out these unpredictable attributes, the memory consumption on both the server side and the client side are acceptable in the current environment.

C. Defense Effectiveness

The goal of the Content Box is to effectively fight against a variety of CSRF attacks. This section will discuss two situations about CSRF attack prevention.

1) CSRF attack in different websites

For a traditional CSRF attack, the victims must visit a malicious website first in order to form a CSRF attack. Consequently, the malicious script in the malicious website forges an HTTP request and the client browser of the victim is forced to send the forged HTTP request. In the Content Box, we can detect this kind of CSRF by examining the HTTP referrer header. In modern client browsers, the HTTP referrer header cannot be modified via JavaScript or HTML. Therefore, the HTTP referrer header is a reliable source to detect all forged HTTP requests originated from different websites. As an example, the flaw of Ebay was attacked as described by Prince [43]. In this scenario, the Content Box can easily block the forged HTTP requests by checking the HTTP referrer header.

There is another similar CSRF attack called the "Login CSRF attack [31]." The concept is to override the cookie of the victims. By our observation, the log-in page is often the most vulnerable page of a website since lack of protection is the main problem. However, the HTTP referrer header is a built-in header, thus the log-in page is protected as well.

2) CSRF attack on the same website

This kind of CSRF attack often cooperates with XSS, and the attacker does not need to set up a website. The attacker can upload malicious scripts to the database of honest websites. When a victim surfs the polluted web page, the client browser of the victim will execute the inserted malicious scripts automatically. The attacker takes advantages of AJAX which can create HTTP requests and customize the HTTP request headers to forge HTTP requests. To prevent multi-stage CSRF attack, the Content Box ensures that UCC is isolated and every HTTP request is tagged with the corresponding label. Once a forged HTTP request is captured, and the multi-stage CSRF attacks can be blocked, respectively.

VII. CONCLUSION

In this paper, we pointed out the fundamental problems of the CSRF attacks and introduced the severity of the CSRF attack on the current social network websites. To cope with the problems, we proposed a novel scheme, Content Box, which can prevent standard and multi-stage CSRF attacks effectively. This approach takes advantage of the built-in methods and the built-in properties to reduce the computation overhead rather than filtering or rewriting the suspicious strings. The web administrator will only need to establish policies for critical services and inserts suspicious contents into iframe tag. This approach also fully utilizes the benefits of Web 2.0; it maximizes the usability of JavaScript and AJAX with minor restriction. The original JavaScript and AJAX

syntaxes and semantics are preserved under the Content Box. For the end users, we proposed a novel design of robust and secure website without the need of altering client browsers. In the design, the users do not need to install additional plug-in or add-on for a specific website. Consequently, the Content Box can prevent CSRF attacks without blocking the interactive content of website.

VIII. ACKNOWLEDGMENT

This work is supported in part by TRUST of UC Berkeley, National Science Council, TWISC, ITRI, III, NCP, iCAST, Chungshan Institute of Science and Technology, Bureau of Investigation, HTC, TrendMicro, Promise Inc., and Chunghwa Telecomm.

REFERENCE

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

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

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

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

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

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

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

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

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

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

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

[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, pp. 661-667, 1997.

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

[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*, pp. 47-60, 2009,.

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

[16] World Wide Web Consortium, "XMLHttpRequest," http://www.w3.org/TR/XMLHttpRequest/, 2009.

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

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

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

[20] Facebook, "Facebook," http://www.facebook.com/.

[21] MySpace, "MySpace," 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*, pp. 554-563, 1992.

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

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

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

[26] 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*, vol. 03, pp. 486–491, 2009.

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

[28] D. Ahmad, "the Confused deputy and the domain hijacker," *IEEE Security and Privacy*, 2008.

[29] H. Volos and H. Teonadi, "Study of security vulnerabilities in Web 2.0," 2007.

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

[31] 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*, pp. 75-88, 2008.

[32] 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, 2006.

[33] 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*, pp. 233–246, 2009.

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

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

[36] 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.

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

[38] W. Zeller and E.W. Felten, "Cross-site request forgeries: Exploitation and Prevention", Technical report, 2008.

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

[40] A. Barth, C. Jackson, and W. Li, "Attacks on JavaScript Mashup Communication," *In Proc. of Web 2.0 Security and Privacy*, 2009.

[41] B. Hoffman, "Ajax security," http://www.spidynamics.com/assets/documents/AJAXdangers.pdf, 2006.

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

[43] Brian Prince, "eBay Security Vulnerabilities Found by Researcher," http://www.eweek.com/c/a/Security/Researcher-Uncovers-eBay-Security-Vulnerabilities-684970/, 2010.

[44] R. Pelizzi, and R. Sekar, "Protection, Usability and Improvements in Reflected XSS Filters," The 7th ACM Symposium on Information, Computer and Communications Security, 2012.

[45] G. Maone, "NoScript," http://noscript.net , 2012.