International Journal of Advance Research in Science and Engineering
Vol. No.4, Special Issue (01), September 2015
www.ijarse.com
IJARSE
ISSN 2319 - 8354

DETECTION AND PREVENTION OF CLICKJACKING AND CROSS SITE REQUEST FORGERY

Vrindamol P[1], Neena V V[2]

[1]*Computer Science and Engineering, Vimal Jyothi Engineering College, Kannur, Kerala, (India)*

[2]*(Asst.Professor, Computer Science and Engineering, Vimal Jyothi Engineering College, Kannur, Kerala, (India)*

ABSTRACT

Web applications are currently a part of day to day life due to their user friendly environment and advancement of technology to provide Internet facilities. Increasing the usage of web applications brought lot of threats with them and these threats are growing continuously. Such type of two main web browser attacks is Clickjacking and CSRF attack. Clickjacking attacks are an important threat on the web. The attacks lure users to click on objects transparently placed in malicious web sites. This type of attack is difficult to detect and thus it is present in most of the existing web applications. Cross site request forgery attack send the malicious request to the legitimate sites. In this survey, I mainly focus on different types of click jacking as well as CSRF vulnerabilities and its existing defenses. When a web page is vulnerable to Clickjacking, it is possible for the attacker to disable CSRF token protection. I have compared various defense mechanisms of Clickjacking and Cross site request forgery to analyze the best defense mechanism. This study will help us to build strong and robust protection mechanism against both Clickjacking and Cross site request forgery.

I. INTRODUCTION

Web applications are important. Web security plays an important role in security of web applications. Nowadays everything is possible through web such as information sharing through social networking and increasing business and delivering service, shopping carts, transactions etc. As one side usage of web is increasing, on the other side attacks on web is also increasing. Providing security in web application is difficult task.

Web applications have evolved from simple collections of static HTML documents to complex, full-edged applications containing hundreds of dynamically generated pages. The combined use of client side and server side scripting allows developers to provide highly sophisticated user interfaces with the various functionalities and look- and- feel. Many web applications have evolved into mesh-ups and aggregation sites that are dynamically constructed by combining together content and functionalities provided by different sources. Combined with the increasing quantity of sensitive information that is now accessible through the web, has brought a corresponding increase in the number of web based attacks and vulnerabilities. [2]. The two main important web browser based attacks are Clickjacking and Cross site request forgery. The idea behind Clickjacking attack and CSRF attacks are similar.

Clickjacking [3] is an act of hijacking user clicks in order to perform undesired actions which are useful for the attacker. Clickjacking is a very common attack through frames. Attacker forces the user to clicks on a malicious page. This is sometimes done by loading the malicious page as a transparent page over a benign page that genuinely needs a click or some input like user login, send email etc. When the user gives the input that was asked, an event is sent to the malicious web page that causes undesirable action to be taken on the user's behalf. With Clickjacking the user actively interact with something, but the action itself can be hijacked by inserting a layer between the user and therefore the legitimate action.

In Cross site request forgery [4] browser doing things on user's without clicking on some event directly. The CSRF attack forces an end user to execute unwanted actions on a web application in which they are presently authenticated. CSRF attacks specifically target state changing requests, not stealing of any data, since the attacker has no way to notice the response to the forged request. If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing email address, and so forth. If the victim has an administrative account, CSRF can compromise the entire web application.

When a web site is prone to Clickjacking, there is a chance for the attacker to disable cross site request forgery (CSRF) token protection that protects against CSRF attacks that trick browsers into doing things without the user's permission or knowledge. Some Clickjacking and CSRF defenses have been proposed and deployed for web browsers, however all have shortcomings. Hence it necessary to possess powerful and robust mechanism to defend against each Clickjacking and CSRF.

II. CLICKJACKING

Clickjacking [3] attacks are an emerging threat on a web. It is additionally referred to as User interface readdress attack or UI Readdressing. It is a malicious technique of tricking a web user into clicking on one thing completely different from what the user perceives. They are clicking on, thus potentially, reveling confidential information or taking management of their computer while clicking on seemingly innocuous web pages.

These types of attacks tempt users to click on objects transparently placed in malicious web pages. The resultant actions of the click function may cause unwanted operations within the legitimate websites without the knowledge of users. Users think that they are clicking on visible buttons whereas they're really performing actions on the hidden page. Example for such sites is online-banking auction sites, web mail services, File sharing sites and Social networking sites.

Malicious web site is in control of the attacker. This page is created in a way so that a transparent IFRAME[5] containing legitimate web sites overlays the content of Malicious site. Since the victim is not aware of the invisible IFRAME, by correctly orienting Target web site over malicious site, an attacker can lure the victim into clicking elements in Target area, while she is below the impression of clicking on malicious web site. A successful Clickjacking attack, for example,Victim deleting all messages from user's web mail inbox, or generating artificial clicks on advertisement banners and performing online money transfer etc. For example a user might receive an email with a link to a video about a news item. for instance a user may however another valid page, say a product page on amazon.com can be hidden on top or underneath the "play" button of news video. The user tries to play the video but actually buys the product from amazon.com. Making frame

transparent is not the only way to mount a Clickjacking attack. A click can also be "stolen" by covering the frame containing the victim page with opaque elements, and then leaving a small hole aligned with the target element on the underlying page. Other types of Clickjacking attack known as Likejacking, Cursor spoofing attack and Whack-a-mole attacks.

Unlike other common web vulnerabilities such as cross site scripting and SQL injection, Clickjacking is not a consequence of a bug in a web application. It is a consequence of a misuse of some HTML/CSS features combined with the way in which the browser allows the user to interact with invisible, or barely visible components' number of techniques to mitigate the Clickjacking problem has been discussed on security related blogs. But they also have some limitations. When a website is vulnerable to Clickjacking it is possible for the attacker to disable cross site request forgery token protection [3].

2.1 Existing Click Jacking Attacks

Existing attacks [9] can be classified according to three ways of forcing the user into issuing input commands out of context:

2.1.1 Compromising Target Display Integrity

1) Hiding the target element: Modern browsers support HTML/CSS styling features that allow attackers to visually hide the target element but still route mouse events to it. The attacker can draw a decoy under the target element by using CSS opacity property and z-index property. Attacker may completely cover the target element with an opaque decoy, but make the decoy Unclickable by setting the CSS property pointer- events: none. A victim's click would then fall through the decoy and land on the invisible target element.

2) Partial overlays: Sometimes, it is possible to visually confuse a victim by obscuring only a part of the target element. For example, attackers could overlay their own information on top of a PayPal checkout iframe to cover the recipient and amount fields while leaving the "Pay" button intact; the victim will thus have incorrect context when clicking on "Pay".

3) Cropping: Wrapping target element in a new iframe and choose CSS position offset properties

2.1.2 Compromising Pointer Integrity

The attacker may violate pointer integrity by displaying a fake cursor icon away from the pointer, known as cursorjacking. This leads victims to misinterpret a click's target, since they will have the wrong perception about the current cursor location. Using the CSS cursor property, an attacker can easily hide the default cursor and programmatically draw a fake cursor elsewhere or alternatively set a custom mouse cursor icon to a deceptive image that has a cursor icon shifted several pixels off the original position.

Another variant of cursor manipulation involves the blinking cursor which indicates keyboard focus. For example, an attacker can embed the target element in a hidden frame, while asking users to type some text into a fake attacker controlled input field. The blinking cursor confuses victims into thinking that they are typing text into the attacker's input field, whereas they are actually interacting with the target element.

2.1.3 Compromising Temporal Integrity

Attacks in the previous two sections manipulated visual context to trick the user into sending input to the wrong UI element. An orthogonal way of achieving the same goal is to manipulate UI elements after the user has

decided to click, but before the actual click occurs. Humans typically require a few hundred milliseconds to react to visual changes and attackers can take advantage of our slow reaction to launch timing attacks.

The double-click attack is based on compromising temporal integrity. The browser do not protect against Framebusting. The double-click attack do Bait and switch that the user do a double click right after the first click. The attacker switches user focus to the Google pop up window under the right before the second click. To predict clicks more effectively, the attacker could ask the victim to repetitively click objects in a malicious game or to double-click on a decoy button, moving the target element over the decoy immediately after the first click.

2.1.4 Whack-a-Mole Attack

The whack-a-mole attack is the hybrid of pointer and temporal integrity. The attacker asks the user to click as fast as possible and suddenly switch Face book like button and gets the profile of the victim. In this attack, attacker ask the user to play a whack-a-mole game and encourage her to score high and earn rewards by clicking on buttons shown at random screen location as fast as possible. Throughout the game, attacker uses a fake cursor to control where the user attention should be. At later point in the game, switch in a Facebook like button at the real cursor's location, tricking user to click on it.

2.1.5 Likejacking Attacks

Likejacking which is a type of Clickjacking attack to hijack the very popular Facebook's like button can be used to cause social degradation in the form of posting unwanted videos and links to a Facebook user's wall without his knowledge. The main idea behind this attack is to hide the like button under the veil of authentic information. Third party widgets such as like button on Face book, tweet button on Twitter, + 1 button on Google Plus, share button and other such type of buttons are all susceptible to c1ickjacking because they can be easily placed in frames and iframes

2.2 Existing Clickjacking Defences

2.2.1 Framebusting:

Elie Bursztein et al. [7] proposed an approach to protect a web page from Clickjacking attack. It consists of a script in each page that should not be framed. The main aim of this technique is to prevent the web page from being loaded in a sub frame. Frame busting is one of the recommended defense mechanism against Clickjacking attack and is also providing security against image based authentication such as the Sign in Seal used by Yahoo. Frame busting code consists of a conditional statement and a counter action that navigates the top page to the correct place. Here a small piece of code is embedded in a page to be protected. If an attempt is made to embed a protected page in an attacker's page the Framebusting code redirects the browser to the original sites so that the victims can see the protected page instead of the attacker page.

Similar to frame busting, the X-Frame-Options header also aims at preventing framing. X-Frame-Options do not rely on JavaScript, but instead it is implemented as a native capability of the web browser. By attaching an X-Frame-Options HTTP response header to an outgoing request, a web server can influence the framing behavior of the corresponding document. It is supported across all the latest browsers. It is not supported in several browsers that are still popular such as Internet Explorer 7 and Firefox 3.6.

2.2.2 Automated Detection of Clickjacking Attacks:

Macro balduzzi et al .[8] introduced an approach for Automated and efficient detection of Clickjacking attacks. When the page has been loaded, extract the coordinates of all the clickable elements in the page and programmatically control the mouse and the keyboard to properly scroll the web page and click on each of those elements. It has two main units. Detection unit and testing unit.

Detection unit is responsible for detecting and logging any Clickjacking attacks that are contained in the web page under analysis. This unit combines two browser plug-in. Both plug-in operate in parallel to analyze the automated clicks. The first plug-in consists of code in order to detect overlapping clickable elements. ClickIDS is the one of the browser plug-in. This intercepts the mouse click event. Detect when multiple clickable elements co-exists and overlays in the region of the page when the user has clicked. When two or more clickable elements of different page overlap at the co-ordinates of the mouse click, ClickIDS report suspicious behavior. ClickIDS precise to identifying attack based on overlapping elements. It is not able to detect attacks based on partially covered elements.

NoScript is a Firefox add-on that provides protection against common security vulnerabilities such as cross-site scripting. It also features a URL access-control mechanism that filters browser-side executable contents such as Java, Adobe Flash. This feature protects users against transparent IFRAME-based attacks. Combination of the two different detection techniques greatly reduces the number of false positives. It intercepts the mouse click events, checks the interactions with the elements of a web page, and detects Clickjacking attacks.

The Testing unit contains a plug-in that extracts the coordinates of the clickable elements rendered on the page. Clickable elements are received from the element extractor. Main component of the testing unit is the Xclick script. It receives the list of URLs to visit and store them, one by one to the browser. Once the page is successfully loaded, the script positions the mouse over each element and clicks on them. Xclick is responsible for manages special elements such as form dropdown boxes which can be rolled up pressing the escape button.

2.2.3 In Context

The root cause of Clickjacking is that an attacker application presents a sensitive UI element of a target application out of context to the user, and hence the user gets tricked to act out of context. L.Huang et al. [6] introduced InContext technique

InContext enforces target display integrity by comparing the OS level screenshot of the area that contains the sensitive element and the bitmap of the sensitive element rendered in isolation at the time of user action. If these two bitmaps are not the same, then the user action is canceled and not delivered to the sensitive element.

This design is resilient to new visual spoofing attack vectors because it uses only the position and dimension information from the browser layout engine to determine what the user sees.

2.2.4 Trueclick

This technique explore the problem of automatically assisting Internet users in detecting malicious trick banners and helping them identify the correct link based on their visual properties such as image size, color, placement on the enclosing web page, whether they contain animation effects, and whether they consistently appear with the same visual properties on consecutive loads of the same web page. Ali Osman Ulusoy et al.[11] implemented a tool called TrueClick, which uses image processing and machine learning techniques to build a classifier based on these features to automatically detect the trick banners on a web page.

TrueClick is implemented as a browser extension that runs on demand when the user visits a file sharing website containing trick banners and clicks on a button to activate the system. Once the analysis of the banner images is complete, TrueClick can either mark the detected trick banners as such, or block them entirely.

2.2.5 CSCP

Ubaid Ur Rehman et al.[10] proposed a browser based solution to detect and prevent Clickjacking attacks. It is referred to as Cursor Spoofing and Clickjacking Prevention. CSCP ensures protection Cursor spoofing attack with high effectiveness and also the Likejacking attacks, other variation of Clickjacking attacks which associate malicious code to Facebook Like buttons. This technique covers the functionality of both the existing extensions and also ensures pointer integrity. Hence it name as cursor spoofing and Clickjacking prevention.

CSCP has the functionality of detecting and preventing Clickjacking attacks on the Facebook social plug -in. When the pointer clicks on like or follow button, pop up appear to inform the user that you click on like or follow

button. When a cursor spoofing is detected on the websites, it displays both the fake and real cursors and also warns the user that the website is compromised. When a user clicks on a hidden button inside an iframe it provides a confirmation box with the message that, you have clicked a hidden button! Are you sure you want to continue? If the user clicks on cancel, the behavior of the hide button will be discarded. Otherwise if user clicks on the yes button then the default actions of the social plug-in take place.

Main drawback of this technique is not providing protection against outside the domain facebook.com. It can detect only Likejacking type attacks.

2.2.6 Proclick

Hisham Haddad et al.[12] proposed a proxy level framework to detect Clickjacking attacks. ProClick examines the content of requests and response pages at the proxy level to detect Clickjacking attacks. Proclick analyzes web pages before being rendered to the browser for detecting Clickjacking attack symptoms. The framework is a client side proxy which can intercept the incoming requests and response pages. It analyze the parameter values of request pages and HTML JavaScript code of response pages and perform systematic checking to decide if attack symptoms are present or not. The framework can be also extended to detect new Clickjacking attacks. It also incurs negligible performance overhead and does not break up the source code of legacy web applications. It does not depend on specific HTTP header or the enabling/disabling of JavaScript at the browser. ProClick is the core module that has the capability to analyze both requests and response pages. It performs a number of checks based on the specified policy for detecting Clickjacking attacks.

The work flow of ProClick is based on four steps:

Step 1) Request analysis: When a request is issued by a browser, ProClick intercepts the request and analyzes it to Identify parameter values containing possible reflected XSS attack payloads.

Step 2) Request sent to website: The intercepted request is forwarded to the remote website. The request might have been modified based on XSS payload checking policy from Step 1.

Step 3) Response page analysis: The response page received from the remote website is analyzed by ProClick before sending it to the client or browser.

Step 4) Response page sent to browser: The sanitized response page is then delivered to the browser after performing page analysis based on specified attack detection policy.

This can detect both client and server side vulnerabilities. This technique can also extended to detect other advanced attacks, which is a promising direction for future research.

III. CROSS SITE REQUEST FORGERY

CSRF [4] occur when a malicious web site causes a user's web browser to perform unwanted action on a trusted site. CSRF is also known as session riding, one click attack and sea-surf. In this types of attack end user is forced to execute unwanted actions on web applications via currently authenticated account or in CSRF website unauthorized commands are transmitted from a user that are websites trusts. These attacks are also known as sleeping giant of web-based vulnerabilities because many sites on Internet fail to protect against them and they also have been ignored by the web development and security communities.

CSRF vulnerabilities are divided into two major categories. These are: Stored and Reflected.

- Stored CSRF: In stored CSRF vulnerabilities the attacker can use the application itself to provide the victim the exploit link or other content which directs the victim browser back into the application and causes attacker controlled actions to be executed as the victim.
- Reflected CSRF: In reflected CSRF vulnerabilities the attacker uses a system outside the application to expose the victim to the exploit link or content. This can be done using a blog, an email message, an instant message, a message board posting, or even a flyer posted in a public place with a victim types in. Reflected CSRF attacks will frequently fail, as users may not be currently logged into the target system when the exploits are tried. The trail from a reflected CSRF attack may be under the control of the attacker, however, and could be deleted once the exploit was completed.

CSRF attack takes the advantage of HTTP protocols functionality to send session cookie for each request to server once user authenticate successfully, which helps server to confirm that the request is coming from authenticated user. CSRF attacker first study the request pattern that is type of request (GET request or POST request), parameters names, type of parameters values etc. Once studied the request's URL pattern deeply, he embed this URL in html tags of web pages or emails. Then attacker forces the authenticated user to execute this request. As user is authenticated browser automatically sends session cookie value with this request, server accepts this request and execute it.

For example, the user logged in to his email account and found email saying that check jobs matching to your profile', If user open this email and follow the link given there, in next tab jobs website page will get open. If attacker also has email account in same site as user and attacker know how change password functionality works, attacker can put the corresponding action URL in some HTML tag.

Another possibility for an attacker to access the honest web site is, Information in the form fields sends to the server by using two methods GET and POST, where GET method generate a request which contain all the information itself in request and it is also visible to the user, so attacker can make use of this easily available information to generate valid request. It was suggested that to use POST instead of GET method to stop this vulnerability. But POST method is also not helped to protect web applications from CSRF attack. Once attacker get all form fields he can embed these fields into his web page, which attacker is going to force the victim to open and can put the JavaScript function which allow form to submit on load event.

3.1 Csrf Defensive Mechanism

3.1.1 Checking Referer Header:

In many cases, when the browser issues an HTTP request, it includes a Referer header that indicates which URL initiated the request. The Referer header distinguishes a same-site request from a cross-site request because the header contains the URL of the site making the request.

A site can defend itself against cross-site request forgery attacks by checking whether the request in question was issued by the site itself. This referrer header parameter can be used by browser to check requests domain on client side before forwarding request to server. So web developers check Referer header to protect applications from CSRF. This can be applied in case of critical operation like password change, amount transfer, purchasing items and changing user privileges etc. This defense mechanism is not possible for cross domain operation.

3.1.2 Secret Validation Token:

One approach to defending against CSRF attacks is to send additional information in each HTTP request that can be used to determine whether the request came from an authorized source. This validation token" should be hard to guess for attacker who does not already have access to the user0s account. If a request is missing a validation token or the token does not match the expected value, the server should reject the request. Secret validation tokens can defend against login CSRF, but developers often forget to implement the defense because, before login, there is no session to which to bind the CSRF token. Adam Barth et al. [13] describe CSRF defense mechanism.

There are number techniques for generating and validating token:

- Session Identifier: The browser's cookie store is designed to prevent unrelated domains from gaining access to each Other's cookies. One common design is to use the user's session identifier as the secret validation token. On every request, the server validates that the token matches the user's session identifier.
- Session-Independent Nonce: Instead of using the user's session identifier, the server can generate a random nonce and store it as a cookie when the user first visits the site. On every request, the server validates that the token matches the value stored in the cookie. For example, the widely used Track issue tracking system implements this technique.
- Session-Dependent Nonce: A refinement of the nonce technique is to store state on the server that binds the users CSRF token value to the user's session identifier. On every request, the server validates that the supplied CSRF token is associated with the user0s session identifier. This approach is used by CSRFx, CSRFGuard and NoForge. But has the disadvantage that the site must maintain a large state table in order to validate the tokens.
- HMAC of Session Identifier: Instead of using server side state to bind the CSRF token to the session identifier, the site can use cryptography to bind the two values.

3.1.3 A Proxy-Based Solution:

A proxy-level framework which placed on the server side between the web server and the target application. This proxy is able to inspect and modify client requests as well as the application's replies to automatically and transparently extend applications. An essential prerequisite for this mechanism is the proxy's

ability to associate a user0s session with a valid token. To this end, the proxy maintains a token table with entries that map session IDs to tokens. By decoupling the proxy from the actual application, the XSRF protection can be offered transparently for all applications.

Request Processing

As a first step, check whether the request contains a session ID or not. If there is no session ID in the request, it is classified as benign. The reason is that since the request does not refer to an existing, authenticated session, it is not able to perform any privileged actions. Thus, it can safely pass the request to the target application. If the request does contain a session ID, then consult the token table to check whether there already exists an entry with a corresponding token. If there is such an entry, require that the request also contains this token. A request that fails to satisfy this condition is classified as an XSRF attack. Then generated a warning message to inform the victim about the attack, together with a link to the application's main page.There is no need to terminate the user's current session when an XSRF attack is detected. After following the link provided in the generated warning message, the user can continue her work normally.

When the request contains a session ID that does not exist in the token table, have to assume that a new session was established.The proxy generates a new, random token and inserts the token, together with the session ID, into the token table. Then the request is passed to the target application.

Reply Processing

The task of the reply processing step is to extend the output of a web application such that a subsequent request of the user contains the correct token. This is achieved in a fashion similar to URL rewriting. Assume that the proxy has to process an output page of the target application containing

The following relative hyperlink:

<a href=0index.php? action=logout0>

LogOut

<=a>

Assume further that the proxy has already determined that the client is authenticated, and that a certain session ID is

in use. In this case, it is necessary to rewrite the hyperlink's URL such that it contains the token associated with this

Session ID:

<a href='index.php? action=logout

& token=990>LogOut

< / a>

When the user follows this link, the mechanism has ensured that the proper token is transmitted. The name of the parameter that stores the token can be chosen arbitrarily, but must not interfere with the names of other parameters used by the target application. The token's value is retrieved from the token table that the proxy maintains. The token table should be freed from stale entries regularly to save memory and CPU time. Token table contain a third column named Timestamp, which indicate the point in time when the corresponding entry was last used. When the time that passed since this point is longer than a configurable session life-time, the entry is removed.

By comparing existing defense mechanism against Clickjacking and CSRF attack it is clear that some technique only based on client side and others server side. There is no technique to detect both client and server side as well as no technique can detect both Clickjacking and Cross site request forgery, which is a promising direction for future research.

IV. CONCLUSION

In this survey paper I discussed Clickjacking attack and CSRF attack vulnerabilities which will help to understand Clickjacking attack and CSRF scenario and causes behind it. Also discussed various Clickjacking and CSRF defensive techniques suggested yet. I compared all the techniques. As per analysis it is found that there are many techniques to defend both the attack but still cannot provide full protection. When a site is vulnerable to Clickjacking it is possible for the attacker to disable the cross site request forgery token protection. Similarly if the target application fails to defend against certain other types of attack such as Clickjacking existing mechanism cannot protect against CSRF attack. Hence Robust and strong protection mechanism is necessary to protect web applications. The proposed technique intended to detect both Clickjacking and Cross site request forgery. Hence a single detection tool can defend both the types' attacks at a time and make browser more secure**.**

REFERENCES

[1]. Marco Balduzzi and Manuel Egele, A Solution for the Automated Detection of Clickjacking Attacks, 2010.
[2]. Dhrubajyoti Pathak Dhruwajita Devi and Sukumar Nandi, Vulnerabilities in web browser 2010
[3]. J. Grossman R. Hansen, Clickjacking, 2008.
[4]. Smeulders AW Gevers T, Journal of advanced computer science and technology, 2010.
[5]. WHATW G, The iframe element, 2010.
[6]. A.Moshchuk L.Huang, Clickjacking: Attacks and defenses, 2012.
[7]. Elie Bursztein and Dan Boneh Gustav Rydstedt, Busting frame busting: a study of Clickjacking vulnerabilities on popular sites, 2010.
[8]. Engin Kirda Marco Balduzzi, Manuel Egele, A solution for the automated detection of Clickjacking attacks, 2010.
[9]. MAONE, Noscript changelog, 2012.
[10]. Waqas Ahmad Khan Ubaid Ur Rehman, on detection and prevention of Clickjacking attack for osns, 2013.
[11]. Engin Kirda Sevtap Duman, Ali Osman Ulusoy, and Trueclick: Automatically distinguishing trick banners from genuine download links, 2014.
[12]. Hisham Haddad Hossain Shahriar, Proclick: A framework for testing Clickjacking attacks in web applications, 2013.
[13]. Adam Barth and Collin Jackson, Robust Defenses for Cross-Site Request Forgery, 2008.