ERI: A New Method for Ensuring Request Integrity

Eryue Zhuang[1,2], Zhenzhou Tian[1,2], Xiaojun Cui[1,2], Jian Li[1,2], Zhiwen Wang[1,2]

[1]MOE Key Lab. for Intelligent Networks and Network Security, Xi'an Jiaotong University
Xian 710049, China

[2]Shaanxi Province Key Lab. of Satellite and Terrestrial Network Tech, Xi'an Jiaotong University
Xian 710049, China

{zhuangeryue, xiaojun526, jianlee}@stu.xjtu.edu.cn, zztian@sei.xjtu.edu.cn, wzw@mail.xjtu.edu.cn

ABSTRACT

A series of requests are performed in fixed order to achieve certain requirements in web applications. The request integrity attack (RIA) is applied to steal users' data and identity, by inducing the users to execute malicious requests that are from untrusted sources and violate the regular order. In this paper, an Ensuring Request Integrity (ERI) method is proposed to prevent two major RIAs: Cross Site Request Forgery (CSRF) and Workflow Attack (WF). The AOP (Aspect-Oriented Programming) is applied to instrument monitors into programs during runtime without modifying the source code. Real-time user-application interactions are captured by jQuery event listening, and tokens are dynamically added to ensure the trustworthy of the source and process of each request. By deploying the ERI on six large open source Web applications, the experimental results show that ERI can ensure request integrity without causing negative impacts to the applications and user experience. Moreover, ERI is capable of monitoring and analyzing the dynamical requests and multiple label issue in Web 2.0.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection; K.6.5 [**Management Of Computing And Information Systems**]: Security and Protection — authentication, invasive software, unauthorized access.

General Terms

Security, Design

Keywords

request integrity; cross site request forgery; workflow attack; aspect oriented programming

1. INTRODUCTION

The Internet has evolved from a simple delivery mechanism for static content to powerful distributed applications, yet the stateless HTTP protocol [1] is still used for remote interactions between users and Web applications. The stateless nature of HPPT protocol requires extra auxiliary means such as Cookie, Session or HTTP authentication information for recording states, so as to associate the continuous requests with a user. Typically, when a new session is established, Web applications will create a Cookie for the browser in the Set-Cookie response header. Then the Cookie will be automatically added to all subsequent requests sent to the application, no matter the request is from the link provided in the application, or from other (maybe untrusted) sources. Yet an application server simply uses Cookie to identify the user, or more specifically the user's browser. That is if the application server receives a request with the Cookie identifying a normal user, it will treat the request as a normal one and perform corresponding operations. This authentication mechanism of Web applications can only guarantee that the request is from a user's browser, but it cannot guarantee that the request is issued or approved by the user.

The loopholes in this simple authentication mechanism can be exploited to conduct a series of attacks to the Web server, of which the Request Integrity Attack (RIA) [2] is a most notorious one, where a malicious user or an external attacker tricks a Web application into processing an unintended (the source or process is not credible) request sequence. Specifically, there are two main kinds of RIA: Cross Site Request Forgery (CSRF) [3] and Workflow Attack (WF) [2]. CSRF is conducted by untrusted requests originating from a malicious site, it is the untrusted request source that leads to the attack. While in the WF attack, an attacker forces the Web application to process an incorrect request sequence, it is the untrusted request process that lead to the attack.

Corresponding to the two main kinds of RIA, one way is to ensure the credibility of the request source. As we know, Web Browsers use the Referer header to identify the request source. Thus the server can isolate a request from untrusted sources by checking its Referer header. Yet simply using Referer header faces the following problems:

- The Referer provided by the browser can be manipulated with client-side attacks. Thus the checking result of a request source using Referer is only reliable on the assumption that the browser is absolutely secure. Yet, numbers of ways have been proposed to successfully tamper the Referer in browsers such as IE6 or Firefox [12].
- The Referer typically contains sensitive information that may offend the privacy of users [10]. Also, many organizations and companies are worried that confidential information of their Intranet get leaked through Referer. Also due to the privacy issues, the vast majority of HTTP requests forbid using Referer header [11].

The other way is to ensure that the request process is reliable. For this kind of techniques, the main idea is to ensure that the user-application interactions always work in accordance with the intended sequence, and the application seldom falls into insecure states that RIA brings. Broadly speaking, existing methods can be

MOBIMEDIA 2016, June 18-19, Xi'an, People's Republic of China
Copyright © 2016 EAI
DOI 10.4108/eai.18-6-2016.2264203

divided into the following two categories: Deterministic Finite Automaton (DFA) [2, 4-6] and Anomaly Detection [7-9].

J.Karthick [4] proposed a system-level design method for Web applications. They use Web DFA model to describe the user-application interactions intended by the developer, and they believe request integrity can always be ensured as long as the application is designed exactly obeying the model. However, it requires the developers to abstract the whole application interactions into a DFA, and manually judges whether an operation is sensitive, which is rather labor-intensive. More importantly, this method can only be applied during the design process, it cannot be applied on pre-existing applications.

Swaddler [7] is a server-side method that achieves the detection of request integrity violations by comparing with a normal pattern library, which is constructed by analyzing internal session variables of a Web application and associating invariants with code blocks. The effectiveness of Swaddler is greatly depend on the accuracy of associating invariants with corresponding code blocks.

Besides, there are two drawbacks in existing methods: one is that they need modifying the source code, and the other is that they can't handle new issues such as dynamically creating request and the multiple label problem in Web 2.0 [13]. For example, the BAYAWAK strategy judges all dynamically created requests as attacks.

This paper proposes a new method called ERI to ensure the request integrity from two perspectives: ensuring trusted source and ensuring trusted process. Also, the AOP (Aspect-Oriented Programming) [14] is applied to dynamically instrument monitors into programs without modifying the source code. Real-time user-application interactions are captured by jQuery [15] event listening, and tokens are dynamically added to ensure the source and process of request is trustworthy.

We summarize the main contributions of this paper as follows:

- A new method called ERI is proposed, which can prevent CSRF and WF attack that violate the request integrity. ERI ensures the intended user-application interaction assumed by the developer while without affecting application usage and user experience.
- Utilizing AOP technique, we achieve real-time monitoring for dynamic user-application interaction without modifying the source code.
- By using the jQuery event handler for monitoring user-application interactions, ERI dynamically adds token that indicates the source page of a request. ERI not only solves the problem that the Referer header method faces, but also can solve the new issues such as dynamically creating requests and multiple labels in Web 2.0.

2. ENSURING REQUEST INTEGRITY

ERI ensures the request integrity from two perspectives, trusted request sources and trusted request process. In brief, ERI achieves the protection by appending to each HTTP request with an additional random number token which can be used to validate the source of the request page. Since the token is not contained in the Cookie, it cannot be forged. More specifically, Fig. 1 depicts how ERI works, which mainly consisting of the following steps:

- ERI uses AOP [16] technique to conduct dynamic monitoring of real-time user-application interactions. It allocates a token which identifies the page once a request comes in, and then inject a jQuery script into the response.
- Through jQuery event listening, we intercept and dynamically add tokens into requests.
- The server confirms that token is carried with the request, and validate whether the request source is credible. Further, ERI will continue to confirm whether request processing is credible or not. Thus only trustable requests can be responded by the server.

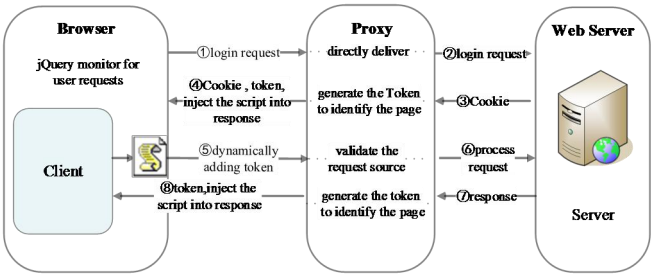

Figure 1. Framework of the ERI protection method.

2.1 Real-time monitoring of Web applications

For the sake of capturing real-time user-application interactions, ERI monitors the entrance methods of each HTTP request. Specifically, the monitoring code is dynamically woven into the application utilizing AOP technique. Thus ERI achieves real-time monitoring of user-application interactions without modifying the source code of the application. By monitoring the process of *doGet* and *doPost* methods, we can obtain various profiler information including the request address (URL), the request parameters, submitted data as well as data that server sends back to the browser etc. ERI is aimed at Java programs, for binary executable files, dynamic instrumentation [20, 21] can be used to dynamically monitor programs' behaviors.

After obtaining the target URL carried in the request during monitoring, we allocate a random number (a token) to the request so as to get the uniquely identified. Note that not all requests will be allocated with a token. Since for HTTP requests like JavaScript requests, CSS requests and various kind of images requests, these resource requests are just for the rendering of pages and are automatically sent by the application. There does not exist interaction between users and these resource requests. Therefore, no tokens will be allocated to these resource requests.

We only allocate token to requests involving user-application interactions. We can recognize these requests by analyzing the target URL, that is, the URL links to valid pages rather than resources. For the token to be allocated, it is 16-bit long, and thus is difficult to guess. The correspondence between the request page and token is stored as a HashMap on the server side.

Further, by adding into the response header of the application Set-Cookie, ERI sends the response page together with its corresponding token back to the local browser. We inject jQuery script into response by directly inserting a HTML script element into the beginning of the document's head section.

2.2 Dynamically adding tokens

To achieve this, we inject into the responses jQuery scripts which will automatically execute after arriving at the browser. With the

jQuery's event listener, we determine whether requests are in the same domain. If so, tokens initially stored locally will be added to these requests together with which are sent to the server. Otherwise, if the requests are in different domains, ERI will not add tokens to the requests to avoid the leakage of the tokens. Besides, the jQuery script will delete Cookies containing corresponding tokens once the tokens are delivered, so as to guarantee the safety of tokens.

Note that, browsers send HTTP requests in a variety of ways, we must ensure that tokens are successfully added to all valid requests. To achieve that, we correctly handle all requests which mainly consist of the following three types: 1) Requests generated by interacting with DOM elements, 2) Requests implicitly generated by HTML tags and 3) Requests created by XMLHttpRequest. Besides, to handle the multiple label issue brought by Web 2.0 browsers, we take into account users' operations such as opening a page through right-clicking from the context menu or dragging link to address bar etc.

RI sends the modified request. In addition, because Web 2.0 browsers can open multiple labels, we have to take the request, that the user opens a page through right-clicking on the mouse from the context menu, into account. The request that caused by dragging the link from the web page to the address bar should also be considered. Because both of them belong to user-application interaction behaviors. After analysis, the common point of the two methods is that users will press the mouse button on the link, so the mousedown event of the href element will register the addGettoken(), as the same process as the click event.

2.3 Ensuring Request Integrity

A request generally consists of a target URL and some parameters in key-value pair form. For operations like database update and deletion, the modification of session state can't be completed without carrying parameters. Therefore, we regard requests with the parameters as the sensitive operations. Request without the parameters won't affect the server's state, and are regarded as non-sensitive operations. ERI only validates the source of sensitive requests.

For each sensitive requests received, ERI validates whether the source page matches the token. If the request carries the correct token, the server will process it. Otherwise, ERI will send a warning information to the user. For the non-sensitive requests, ERI would simply deliver it to the server for processing.

Validating whether requests carry the correct tokens ensure trusted request source. Yet, we must further ensure the trusted request process. To guarantee that, we should firstly validate whether request source is trusted, after that we further judge whether the target request is the key sensitive operation or not. If it is a key sensitive operation, we will validate whether the source page is the previous operation which the application formulates. Only if it is, the request will be passed to the server. If the key sensitive request does not carry the token of the previous request, the server will not handle it. In such a way, ERI guarantees that requests be processed in accordance with the intended order anticipated by the web application.

3. EXPERIMENTAL EVALUATION

We validated the effectiveness of ERI by deploying it to six open-source Java Web applications. The experiments are performed on a Windows machine equipped with Intel® Core ™ i7-2600 3.40GHz CPU and 4.0GB RAM. The six experimental subjects includes: WebGoat[1], Jpetstore[2], CRM[3], JForum[4], JAVAPMS[5] and JEECMS[6], which all have RIA vulnerabilities.

3.1 Compatible Test

In this experiment, we firstly evaluated whether ERI has any negative impact on the regular use of applications. That is, we must ensure that the deployment of ERI to each experimental subject will not bother the functionality of each application.

In specific, we use different browsers including Google Chrome, Firefox and IE to test the compatibility. After deploying ERI, we test the core functions for each application. To achieve that, we do a full coverage test including initiating the GET and POST requests, trying different methods of opening a new page such as just click, right-click and opening a new label, dragging the link to address bar etc. as well as trying multiple methods of navigating to the next step such as clicking the pictures, buttons, links in the browser. These tests show that the ERI brings no damage to the normal functionalities of the applications.

3.2 Prevention Experiment

These six applications all have RIA vulnerabilities. One type is the CSRF vulnerability, based on which attackers can conduct a series of spiteful actions. Specifically, for the experimental subjects, an attacker can: simulate virtual transfers in WebGoat; deceive innocent users to add items to the shopping cart in Jpetstore; delete customer information, announcements etc. in CRM; delete users' inbox mail in JForum and even trick administrator in JAVAPMS.

Two of the applications also exist the WF vulnerability, another type of RIA vulnerability. By conducting straightforward workflow violation attack, an attacker can complete the login operation by directly entering the URL in address bar without accessing the login page in JForum and Jpetstore. Besides, in Jpetstore's purchasing workflow, an attacker could directly submit orders without performing any shopping actions explicitly such as adding item to shopping cart.

Table 1. ERI for preventing RIAs of the applications

Web application	version	CSRF	WF	Attacks Eliminated
WebGoat	5.4	Yes		Yes
Jpetstore	5.0	Yes	Yes	Yes
CRM	1.0	Yes		Yes
JForum	2.1.8	Yes	Yes	Yes
JAVAPMS	1.3	Yes		Yes
JEECMS	6.0	Yes		Yes

[1] http://code.google.com/p/webgoat/

[2] http://sourceforge.net/projects/ibatisjetstore/

[3] https://code.google.com/p/JavaScriptpcrm/

[4] http://www/jforum.net/

[5] http://www.javapms.com/

[6] http://www.jeecms.com/

Table 1 summarizes the results, where the columns give the version of each experimental subjects, the type of RIAs. And as it shown by the last column, ERI can prevent all RIAs.

3.3 Performance Overhead

The experiments conducted above show that ERI can ensure request integrity without affecting the original functions of the applications. On the other hand, we must assure that the ERI will not introduce obvious performance degradation once deployed. To evaluate the overhead, we compare the average response time of user requests before and after the deployment of ERI.

Specifically, we utilize JMeter [17], a load testing tool, to drive the performance evaluation process. For each application, we perform a series of operations and generate JMeter scripts. For example, a script can record the whole shopping process from querying the goods to the payment in JPetStore. Then with the scripts, we modify parameters such as setting the number of concurrent requests, the times of request repetitions etc. Finally, we utilize these scripts to drive the execution of each application, and calculate the average response time. As summarized in Table 2 the experimental results, the performance overhead ranges between 3.12% and 10% depending on the size of the web applications. The overhead is acceptable.

Table 2. Performance Overhead of ERI

Web application	Application Response (msec)	ERI Overhead (msec)	Percent Overhead (%)
WebGoat	400	40	10.0
Jpetstore	1340	60	4.48
CRM	4680	120	2.56
JForum	4740	360	7.59
JAVAPMS	22420	700	3.12
JEECMS	12300	700	5.69

4. CONCLUSION

In this paper, we proposed ERI for ensuring request integrity from two perspectives — trusted source and trusted process. Unlike existing methods, ERI does not need modifying the source code of an application. Besides, by utilizing jQuery event handler to dynamically add tokens, it is capable of handling the dynamically created requests and multiple label issues brought by Web 2.0. The deployment of ERI on six large web applications show that it can effectively prevent RIAs, meanwhile brings little side effect to the applications.

5. ACKNOWLEDGMENT

The research was supported in part by National Science Foundation of China (91118005, 91218301, 61221063, 61428206, 61203174, 91418205, 61472318, 1500365), Ministry of Education Innovation Research Team (IRT13035), Key Projects in National Science and Technology Pillar Program (2013BAK09B01).

6. REFERENCES

[1] Fielding R, Gettys J, Mogul J, et al. Hypertext transfer protocol–HTTP/1.1. RFC 2616, June, 1999.

[2] Jayaraman K, Lewandowski G, Talaga PG, et al. Enforcing request integrity in web applications. Data and Applications Security and Privacy XXIV. *Springer*, 2010: 225-240.

[3] Watkins P. Cross-site request forgeries[J]. *Bugtraq mailing list*, 2001.

[4] Jayaraman K, Talaga PG, Lewandowski G, et al. Modeling user interactions for (fun and) profit: preventing request forgery attacks on web applications[C]: *ACM*, 2009: 16.

[5] Andrews AA, Offutt J, Alexander RT. Testing web applications by modeling with FSMs[J]. *Software & Systems Modeling*, 2005, 4 (3): 326-345.

[6] Yuen S, Kato K, Kato D, et al. Web Automata: A Behavioral Model of Web applications based on the MVC model[J]. 2005, 22 (2): 44-57.

[7] Cova M, Balzarotti D, Felmetsger V, et al. Swaddler: An approach for the anomaly-based detection of state violations in web applications[C]: *Springer*, 2007: 63-86.

[8] Kruegel C, Vigna G. Anomaly detection of web-based attacks[C]: *ACM*, 2003: 251-261.

[9] Kruegel C, Vigna G, Robertson W. A multi-model approach to the detection of web-based attacks[J]. *Computer Networks*, 2005, 48 (5): 717-738.

[10] Johnson A. The Referer header, intranets and privacy, February 2007. 2009.

[11] Harold ER. Privacy tip# 3: Block Referer headers in Firefox, October 2006.

[12] Klein A. Exploiting the XMLHttpRequest object in IE—Referrer spoofing and a lot more..., September 2005. 2009.

[13] Noureddine AA, Damodaran M. Security in web 2.0 application development[C]: *ACM*, 2008: 681-685.

[14] Kiczales G, Lamping J, Mendhekar A, et al. Aspect-oriented programming. ECOOP'97—Object-oriented programming. *Springer*, 1997: 220-242.

[15] Resig J. jquery: The write less, do more, javascript library[J]. Disponıvel em http://jquery com/, Acesso em, 2009, 18 (04): 2009.

[16] Kiczales G, Hilsdale E, Hugunin J, et al. An overview of AspectJ. ECOOP 2001—Object-Oriented Programming. *Springer*, 2001: 327-354

[17] Halili EH. Apache JMeter: A practical beginner's guide to automated testing and performance measurement for your websites[M]: Packt Publishing Ltd, 2008.

[18] Jovanovic N, Kirda E, Kruegel C. Preventing cross site request forgery attacks[C]: *IEEE*, 2006: 1-10.

[19] Johns M, Winter J. RequestRodeo: Client side protection against session riding[C], 2006.

[20] Tian Z, Zheng Q, Liu T, et al. Plagiarism detection for multithreaded software based on thread-aware software birthmarks[C]. *Proceedings of the 22nd International Conference on Program Comprehension*. ACM, 2014: 304-313.

[21] Tian Z, Zheng Q, Liu T, et al. Software Plagiarism Detection with Birthmarks Based on Dynamic Key Instruction Sequences[J]. *Software Engineering, IEEE Transactions on*, 2015, 41(12): 1217-1235.