Re-designing the Web's Access Control System (Extended Abstract)*

Wenliang Du, Xi Tan, Tongbo Luo, Karthick Jayaraman, and Zutao Zhu

Department of Electrical Engineering and Computer Science
Syracuse University, Syracuse, New York, 13244, USA
Email: wedu@syr.edu Tel: +1 315 443-9180

Abstract. The Web is playing a very important role in our lives, and is becoming an essential element of the computing infrastructure. With such a glory come the attacks–the Web has become criminals' preferred targets. Web-based vulnerabilities now outnumber traditional computer security concerns. Although various security solutions have been proposed to address the problems on the Web, few have addressed the root causes of why web applications are so vulnerable to these many attacks. We believe that the Web's current access control models are fundamentally inadequate to satisfy the protection needs of today's web, and they need to be redesigned. In this extended abstract, we explain our position, and summarize our efforts in redesigning the Web's access control systems.

Key Words: web security; access control model.

1 Introduction

The Web is playing a very important role in our lives, and is becoming an essential element of the computing infrastructure. Because of its ubiquity, the Web has become attackers' preferred targets. Web-based vulnerabilities now outnumber traditional computer security concerns [2, 4]. SQL injection, cross-site scripting (XSS), and cross-site request forgery are among the most common attacks on web applications. A recent report shows that over 80 percent of websites *have had* at least one serious vulnerability, and the average number of serious vulnerabilities per website is 16.7 [26].

Attacks on the Web are quite unique, compared to the attacks on the traditional computer systems and networks. From the top 10 list of web attacks recently release by OWASP [19], we can tell that these attacks, to a large degree, are attributed to the unique architecture of web applications. In general, the most common structure for web applications is three-tiered [20]: presentation, application, and storage. The web browser belongs to the first tier, presentation. The web server, using technologies like PHP, ASP, ASP.NET, etc., is the middle tier, which controls the application logic. The database is in the storage tier. Therefore, a typical web application consists of three major components: contents (static and dynamic, such as Javascript code) for the presentation tier, code for the application tier, and interactions with the database.

Various security solutions have been proposed to address the problems on the Web [1, 5, 6, 10, 11, 14, 15, 17, 20, 21]; although some of them are quite effective in defending

* This work was supported by Award No. 1017771 from the US National Science Foundation.

against certain specific type of attacks, few have answered the questions "why is the Web so vulnerable to these many attacks" and "what are the root causes of these problems". If we do not address the root causes, we may be able to address some known problems today, but more and more problems may arise in the future, as the Web is still evolving and new features are being introduced from time to time. We need to study the fundamental problems of why web applications are so vulnerable, and develop solutions to address these fundamental problems, instead of developing point solutions to fix each specific attack.

Most of the vulnerabilities appear to be caused by the mistakes in the programs, but, when we look deeper and think about why the developers make such mistakes, we realize that the real problem is the underlying access control architecture: because of the inadequacy of the access control support from the underlying architecture, developers are forced to implement additional access control in their programs. History has told us that asking average developers to implement access control is dangerous, and that being able to build software systems does not necessarily mean being able to build the security part correctly.

Let us look retrospectively at how the access control in operating systems has been evolved to counter the ever-increasing threats. We can see a clear trend: access control has evolved from the simple access control list, to capability-based access control in Linux [8] and Solaris [23], and to the support of more complicated Mandatory Access Control (MAC) models in SELinux [18] and Windows Vista [3]. These sophisticated access control mechanisms free application developers from building all the access control in their own applications; they can rely on the operating system to do most of the access control work.

Unfortunately, web application developers do not have such a good luck, because the access control mechanisms in the web architecture are quite rudimentary. Although the Web has been evolved quite significantly, with new features being added and new types of data incorporated, the underlying protection model is basically the same as that in the early days, and it has become much insufficient for the Web today. To make up for the insufficiency of the underlying protection model, application developers have to include a lot of access control logics in their programs. This is the exact task that the operating systems strive to free developers from. While much work has been done to secure web applications without changing the fundamental access control model, we take a bold and significantly different position in our research:

> **Our position:** We believe that the current access control models of the web architecture are fundamentally inadequate for the Web; they need to be redesigned to address the protection needs of the current Web. A well-designed access control model can simplify application developers' tasks by enforcing much of the access control within the model, freeing developers from such a complicated and error-prone task.

To understand our position, we need to understand the access control architecture underlying web applications. Conceptually, the access control in web applications can be divided into two parts: browser-side and server-side access control. We will discuss them in the next section.

2 Current Access Control in the Web

2.1 Browser-side access control

Web applications have evolved to become highly interactive applications that execute on both the server and client. As a result, web pages in modern applications are no longer simple documents–they now comprise highly dynamic contents that interact with each other. In some sense, a web page has now become a "system": the dynamic contents are programs running in the system, and they interact with users, access other contents both on the web page and in the hosting browser, invoke the browser APIs, and interact with the programs on the server side. To provide security, web browsers adopt an access control model called *Same Origin Policy (SOP)*. SOP prevents the active contents belonging to one origin from accessing the contents belonging to another origin, but it gives all the active contents from the same origin the same privileges.

Unfortunately, today's web pages no longer draw contents from a single source; contents are now derived from several sources with varying levels of trustworthiness. Contents may be included by the application itself, derived from user-supplied text, or from partially trusted third parties. Web applications merge these contents into web pages, which are then sent to users' browsers at their requests. During parsing, rendering, and execution inside the browser, entities (dynamic and static) in web pages can both act on other entities or be acted upon—in classic security parlance, they can be instantiated as both principals and objects. These principals and objects are only as trustworthy as the sources from which they originate.

With the SOP model, all these contents have the same privileges, because once embedded into a web page, from the browser's perspective, they are indeed from the same origin, and will be treated the same. This is a limitation of the SOP model. Since SOP cannot enforce access control based on contents' actual originating sources, web applications have to implement the control at the server side, even though the access actually takes place at the browser side. The goal of this access control approach is to conduct checking and filtering at the server side before merging the contents into web pages, thereby preventing specific, known attacks from even initiating an action within the generated web pages. For example, to defeat the cross-site scripting attack, one can filter out the code from the contents that are from untrusted sources.

Conducting browser-side access control at the server side has a number of limitations. First, doing the filtering and validation has proven to be difficult; many vulnerabilities are caused by the errors in such a process [7, 9, 12]. For example, despite the fact that `Myspace` had implemented many filtering rules, the Samy worms still found the ways to inject unauthorized Javascript code into users' profiles [13]. Second, if web applications need to run some third-party code (e.g. advertisement and client-side extensions) on a web page, but want to put a limitation on the code (e.g. disallow the access to cookies), it will be difficult, if possible at all, for input validation and filtering to achieve this goal on the server side. In a recent event (September 2009), an unknown person or group, posing as an advertiser, sneaked a rogue advertisement onto New York Times' pages, and successfully compromised the integrity of the publisher's web application using a malicious Javascript program [25]. Third, since the accesses actually take place at the browser side, the server side is fundamentally the wrong place to control

these accesses. Access control should be conducted at the run time, when the access is already initiated; this way, we will have all the contexts for access control, including principals, objects, and the condition of the environment.

Therefore, we strongly believe that the browser-side access control should be put back to its proper location, namely, in browsers. This cannot be achieved with the current SOP access control model; a new access control model needs to be developed for web browsers.

2.2 Server-side access control

On the server side, access control is primarily based on sessions. When a user logs into a web application, the server creates a dedicated session for this user, separating him/her from the other users. Sessions are implemented using session cookies; as long as a request carries a session cookie, it will be given all the privileges associated with that session. *Namely, within each session, all requests are given the same privileges, regardless of whether they are initiated by first-party or third-party contents, from client-side or server-side extensions, or from another origin.* We refer to this access control as the "same-session" policy.

Such a single level of granularity, being sufficient for the earlier day's Web, becomes inadequate to address the protection needs of today's Web. The Web, initially designed for primarily serving static contents, has now evolved into a quite dynamic system, consisting of contents and requests from multiple sources, some more trustworthy than others. For example, nowadays, many web applications include *client-side extensions*, i.e., they include links to third-party code or directly include third-party code in their web pages. Examples of client-side extensions include advertisements, Facebook applications, iGoogle's gadgets, etc. Their contents, containing JavaScript code, can be very dangerous if they are vulnerable or malicious,

Unfortunately, the current session-based access control at the web server cannot treat these third-party contents differently. In the current access control systems, it is very difficult to allow the requests from the same web page to access the same session, while preventing some of them from invoking certain server-side services. To achieve such a distinction, applications have to implement their own ad hoc protection logic, such as asking users to confirm their actions, embedding tokens in hidden fields, etc.

The fundamental cause of the above problem is the granularity of a session: it is too coarse. The Web has become more and more complicated, and its client-side contents are no longer uniformly trusted, so requests initiated by these contents are not uniformly trusted either. Therefore, giving all the requests within the same session the same privileges cannot satisfy the protection needs of today's Web anymore. In order not to ask application developers to bear the complete responsibility of implementing those protection needs, we need a better server-side access control system.

3 Our Approaches

Our approach is inspired by the access control in operating systems. Operating systems consider the implementation of access control as their own responsibility, instead of the

responsibility of their applications. This is for security reasons, because OS needs to guarantee that all the accesses are mediated; relying on applications to enforce access control simply cannot achieve this goal. Unfortunately, in web applications, because of the lack of appropriate access control models, web applications have to implement their own access control mechanisms, which tend to be error prone: if they miss some places, loopholes may be created.

To satisfy the needs of access control, most operating systems have built in some basic access control models, such as the ACL model in most OSes, an integrity-focused MAC model since Windows Vista [3], and a fine grained MAC model in SELinux [18]. With these models, user applications do not need to worry about implementing some of the access controls if they can be covered by the models. For example, if an application system's protection needs can be satisfied by the underlying ACL model, it only needs to properly *configure* all the objects in the system, and then relies on the operating system to enforce the access control. If an application system needs to enforce a specific MAC policy in SELinux, it only needs to *configure* its system, and then lets SELinux to enforce the access control; the configuration in this case includes setting up the security policies and labeling the subjects and objects.

The benefit of replacing implementation with configuration can be summarized briefly in the following: First, from the implementation perspective, configuring a system is easier than implementing a system, and is thus less error-prone (although errors are still possible). Second, from the verification perspective, because configuration is usually defined based on logics that are much simpler than programming logics, verifying configuration is also much easier than verifying programs. Third, from the error-resistance perspective, configuration is safer: any missing configuration can fall back to a safe default; however, there is no "safe default" if an access control checking is missing. When a web application has over 1000 security checks, missing a few checks is not uncommon [27]. Fourth, configuration allows web applications to put the access control in the place where the access actually takes place.

Motivated by the successful practice in operating systems and the benefit of configuration, *we set out to investigate whether we can develop a better access control system for the Web, such that we can take some of the access control enforcement logic out of web applications, and replace them with configuration, a much easier task*. The enforcement will be done by the access control system that we develop for browsers, servers, and databases. We summarize our ongoing efforts in the following.

Browser-side access control: We have developed two access control models for web browsers: Escudo [11] and Contego [16]. Escudo proposes a ring access control model for web browsers. This model allows web applications to put webpage contents in different rings, based on their trustworthiness: Elements accessible only to more trustworthy principals or from more trusted sources are placed in higher privileged rings. Ring assignments are carried out at the server side, because only the server-side code knows how trustworthy the contents are. Assigning ring labels to contents is called "configuration", and once a web page is "configured", the browser can enforce access control based on the configuration and Escudo's security policies: contents in the lower-privileged rings cannot access the contents in the higher-privileged rings. We implemented Escudo in a browser called Lobo [22].

To provide an even finer granularity, we have developed Contego, a capability-based access control for web browsers. Contego divides the action privileges (e.g. accessing cookies, sending AJAX requests, etc) into small "tokens" (called capabilities). A principal needs to possess the corresponding tokens if it wants to perform certain actions. For example, a Javascript code within a web page will not be able to send AJAX requests if it is not assigned the AJAX-request token. Using these fine-grained capabilities, web applications can assign the least amount of privileges to principals. We implemented Contego in the Google Chrome browser.

Server-side access control: We have developed a fine-grained server-side access control system, which can assign different privileges to the requests in the same session, based on their trustworthiness. The new access control system is called Scuta [24], which is a backward-compatible access control system for web application servers. Extending Escudo's ring model to the server, Scuta labels server-side data (e.g. tables in database) and programs (functions, classes, methods, or files) with rings, based on their protection needs. Programs in a lower-privileged ring cannot access data or code in a higher-privileged ring.

Scuta divides a session into multiple *subsessions*, each mapped to a different ring. Requests from a more trustworthy region in a web page belong to a more privileged subsession. Requests belonging to subsession k are only allowed to access the server-side programs and data in ring k and above (numerically). With the subsession and ring mechanisms, server-side programs can treat the requests in the same session differently, based on the trustworthiness of their initiators, and thus provide access control at a finer granularity. Subsessions in Scuta correspond to the rings in Escudo, i.e., requests initiated from Escudo ring k in a web page is considered as belonging to subsession k, and can thus access the corresponding server-side resources.

To demonstrate the effectiveness of Scuta, we have implemented Scuta in PHP, a widely adopted platform for web applications. We have conducted comprehensive case studies to demonstrate how Scuta can be used to satisfy the diversified protection needs in web applications.

4 Summary

We strongly believe that the access control systems in the current Web infrastructure is fundamentally inadequate to satisfy the protection needs of today's Web, and they have, directly and indirectly, contributed to the dire situation in web applications. It is time to think about whether we can design a better and backward-compatible access control system, instead of developing fixes to patch the existing one in order to defeat certain specific attacks. The web technology is still evolving, so a good design should not only be able to satisfy today's needs, it should also be extensible to satisfy the unknown protection needs that will inevitably come up during the technology evolution. In this extended abstract, we have summarized our pursuit in building a better access control system for the Web.

5 Acknowledgment

Several other people have also participated in this research, including Amit Bose, Steve Chapin, Tzvetan Devnaliev, Hao Hao, Apoorva Iyer, Balamurugan Rajagopalan, Karthick Soundararaj, Shaonan Wang, and Yifei Wang. We would like to acknowledge their contributions.

References

1. Caja. http://code.google.com/p/google-caja/.
2. S. Christey and R. A. Martin. Vulnerability type distributions in cve (version 1.1). MITRE Corporation. http://cwe.mitre.org/documents/vuln-trends/index.html, 2007.
3. M. Conover. Analysis of the windows vista security model. Symantec Corporation, http://www.symantec.com/avcenter/reference/Windows_Vista_Security_Model_Analysis.pdf, 2007.
4. Symantec Corp. Symantec internet security threat report: Trends for july-december 2007 (executive summary). Page 1–2, 2008.
5. Douglas Crockford. ADSafe. http://www.adsafe.org.
6. M. Dalton, C. Kozyrakis, and N. Zeldovich. Nemesis: Preventing authentication & access control vulnerabilities in web applications. In *Proceedings of the Eighteenth Usenix Security Symposium (Usenix Security)*, Montreal, Canada, 2009.
7. Jeremiah Grossman. Cross-site scripting worms and viruses. The impending threat and the best defense. http://www.whitehatsec.com/downloads/WHXSSThreats.pdf.
8. S. E. Hallyn and A. G. Morgan. Linux capabilities: making them work. http://ols.fedoraproject.org/OLS/Reprints-2008/hallyn-reprint.pdf, 2008.
9. Robert Hansen. XSS cheat sheet. http://ha.ckers.org/xss.html.
10. Collin Jackson, Andrew Bortz, Dan Boneh, and John C. Mitchell. Protecting browser state from web privacy attacks. In *WWW 2006*.
11. K. Jayaraman, W. Du, B. Rajagopalan, and S. J. Chapin. Escudo: A fine-grained protection model for web browsers. In *Proceedings of the 30th International Conference on Distributed Computing Systems (ICDCS)*, Genoa, Italy, June 21-25 2010.
12. Samy Kamkar. The samy worm story. http://namb.la/popular/, 2005.
13. Samy Kamkar. Technical explanation of the myspace worm. http://namb.la/popular/tech.html, 2005.
14. Chris Karlof, Umesh Shankar, J. D. Tygar, and David Wagner. Dynamic pharming attacks and locked same-origin policies for web browsers. In *CCS 2007*.
15. Benjamin Livshits and Úlfar Erlingsson. Using web application construction frameworks to protect against code injection attacks. In *PLAS 2007*.
16. T. Luo and W. Du. Contego: Capability-based access control for web browsers. In *Proceedings of the 4th International Conference on Trust and Trustworthy Computing*, Pittsburgh, PA, 2011.
17. Leo A. Meyerovich and V. Benjamin Livshits. Conscript: Specifying and enforcing fine-grained security policies for javascript in the browser. In *IEEE Symposium on Security and Privacy*, pages 481–496, 2010.
18. National Security Agency. Security-Enhanced Liunx. Available at http://www.nsa.gov/selinux/.

19. OWASP. The ten most critical web application security risks. http://www.owasp.org/index.php/File:OWASP_T10_-_2010_rc1.pdf, 2010.
20. Bryan Parno, Jonathan M. McCune, Dan Wendlandt, David G. Andersen, and Adrian Perrig. CLAMP: Practical prevention of large-scale data leaks. In *Proc. IEEE Symposium on Security and Privacy*, Oakland, CA, May 2009.
21. K. Patil, X. Dong, X. Li, Z. Liang, and X. Jiang. Towards fine-grained access control in javascript contexts. In *Proceedings of the 31st International Conference on Distributed Computing Systems (ICDCS)*, Minneapolis, Minnesota, USA, June 20-24 2011.
22. Jose Solorzano. The Lobo Project. http://lobobrowser.org/.
23. SUN Microsystems, Inc. White paper: Trusted Solaris 8 operating environment. Available at http://www.sun.com/software/whitepapers/wp-ts8/ts8-wp.pdf.
24. X. Tan, W. Du, T. Luo, and K. Soundararaj. SCUTA: A server-side access control system for web applications. Syracuse University Technical Report, 2011.
25. Ashlee Vance. Times web ads show security breach. http://www.nytimes.com/2009/09/15/technology/internet/15adco.html.
26. WhiteHat Security. Whitehat website security statistic report, 10th edition, 2010.
27. A. Yip, X. Wang, N. Zeldovich, and M. F. Kaashoek. Improving application security with data flow assertions. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles*, Big Sky, MT, October 11-14 2009.