Western Kentucky University
TopSCHOLAR®

Masters Theses & Specialist Projects | Graduate School

5-25-2012

Dynamic Scoping for Browser Based Access Control System

Vinaykumar Nadipelly
Western Kentucky University, vinaykumar.nadipelly408@topper.wku.edu

Follow this and additional works at: http://digitalcommons.wku.edu/theses

Part of the Databases and Information Systems Commons

Recommended Citation
Nadipelly, Vinaykumar, "Dynamic Scoping for Browser Based Access Control System" (2012). *Masters Theses & Specialist Projects.* Paper 1149.
http://digitalcommons.wku.edu/theses/1149

This Thesis is brought to you for free and open access by TopSCHOLAR®. It has been accepted for inclusion in Masters Theses & Specialist Projects by an authorized administrator of TopSCHOLAR®. For more information, please contact connie.foster@wku.edu.

DYNAMIC SCOPING FOR BROWSER BASED ACCESS CONTROL SYSTEM

A Thesis
Presented to
The Faculty of the Department of Mathematics and Computer Science
Western Kentucky University
Bowling Green, Kentucky

In Partial Fulfillment
Of the Requirements for the Degree
Master of Science

By
Vinaykumar Nadipelly

May 2012

DYNAMIC SCOPING FOR BROWSER BASED ACCESS CONTROL SYSTEM

Date Recommended April 20 2012

Dr. Guangming Xing, Director of Thesis

Dr. Huanjing Wang

Dr. Qi Li

18-May-2012

Dean, Graduate Studies and Research Date

ACKNOWLEDGMENTS

I would like to express my gratitude to all those who gave the possibility to complete this thesis. I would like to thank my thesis advisor, Dr. Guangming Xing for his guidance, and immense trust and patience he has over me.

I would also like to thank Dr. Huanjing Wang and Dr. Qi Li for their valuable time and suggestions on my thesis.

Finally, I would like to thank my family for their constant love, support and motivation.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION 1

1.1 Nature of the web 2

1.2 Contents of the web 3

1.2.1 Trusted contents 4

1.2.2 Untrusted contents 4

1.2.3 Partially trusted contents 4

1.3 Attacks on web 5

1.3.1 Access control system 7

1.3.2 Web components 7

CHAPTER 2: THE PURPOSE FOR PROTECTION ENHANCEMENT 9

2.1 Same origin policy 9

2.1.1 Cross Site Scripting (XXS) 11

2.2 Same session policy 13

2.2.1 Cross Site Request Forgery (CSRF) 13

2.3 Failure to support design principles 15

2.3.1 Separation of privilege 15

2.3.2 Least privilege 16

CHAPTER 3: RELATED WORK 17

CHAPTER 4: TWO-WAY SECURITY MODEL 20

CHAPTER 5: EFFECTUATION OF TWO-WAY SECURITY MODEL 27

5.1 Lobo Architecture 27

5.1.1 User Interface 27

5.1.2 Browser Engine 28

5.1.3 Cobra HTML Rendering Engine 29

5.1.4 Rhino JavaScript Interpreter 29

5.1.5 Object Wrapper 30

5.2 Identifying subsystems of the Lobo browser architecture for implementation 30

5.3 Two-way security model implementation 32

5.3.1 Extracting and Tracking security groups 32

5.3.2 Enforcing access control policy 33

CHAPTER 6: CONCLUSION 36

BIBLIOGRAPHY 37

LIST OF FIGURES

Figure 1: Example for different types of web contents..... 5

Figure 2: Without same origin policy 10

Figure 3: With same origin policy 11

Figure 4: Illustrate XSS Attack..... 12

Figure 5: Illustrate CSRF Attack 14

Figure 6: Two-way security model 21

Figure 7: Difference between static and dynamic scoping. 24

Figure 8: Complete example demonstrates working of a two-way security model..... 26

Figure 9: Architecture of Lobo browser 28

DYNAMIC SCOPING FOR BROWSER BASED ACCESS CONTROL SYSTEM

Vinaykumar Nadipelly May 2012 39 Pages

Directed by: Dr. Guangming Xing, Dr. Huanjing Wang and Dr. Qi Li

Department of Mathematics and Computer Science Western Kentucky University

We have inorganically increased the use of web applications to the point of using them for almost everything and making them an essential part of our everyday lives. As a result, the enhancement of privacy and security policies for the web applications is becoming increasingly essential. The importance and stateless nature of the web infrastructure made the web a preferred target of attacks. The current web access control system is a reason behind the victory of attacks. The current web consists of two major components, the browser and the server, where the effective access control system needs to be implemented. In terms of an access control system, the current web has adopted the inadequate same origin policy and same session policy for the browser and server, respectively. The current web access control system policies are sufficient for the earlier day's web, which became inadequate to address the protection needs of today's web.

In order to protect the web application from un-trusted contents, we provide an enhanced browser based access control system by enabling the dynamic scoping. Our security model for the browser will allow the client and trusted web application contents to share a common library and protect web contents from each other, while they still get executed at different trust levels. We have implemented a working model of an enhanced browser based access control system in Java, under the Lobo browser.

CHAPTER 1: INTRODUCTION

The Internet is a global system of interconnected computer networks that uses the standard Internet protocol suite to serve billions of users worldwide [29]. It allows us to provide easy and efficient communication between any place in the world, work from remote locations, locate and retrieve the useful information within seconds, and access services and entertainment via World Wide Web (Web). It is a very common mistake for most people to treat the terms “Internet” and ‘World Wide Web” as interchangeable. The words “Internet” and “World Wide Web” are not the same thing but related things. The Internet is a global network of networks. In contrast, the Web is one of the applications that run on the Internet. For example, the Internet is a restaurant and the Web is the most popular dish on the menu.

The volume of traffic moving over the Internet, as well as corporate networks, is expanding exponentially every day. It is estimated that there are over 2.26 billion people worldwide with Internet access as of December 31, 2011 [1]. While communication companies contend frantically to bring faster transmissions into homes, and with the Internet evolving to deliver new forms of services and entertainment, many experts predict that the best is yet to come.

The Web has a tremendous impact on our personal lives, through which large volumes of personal and business communications are taking place. It has now evolved to account for large portions of corporate revenue. There was tremendous progress in its development since the Web was invented. The current Web is no longer a platform for

simple static pages; it has evolved to highly dynamic and interactive ones. The Web is indispensable in education, security, modern commerce, entertainment, and social interaction. It became a complex delivery platform for sophisticated, distributed applications with multifaceted security requirements. Analysts are constantly trying to find out the number of web pages available on the Web. However, it is quite impossible to analyze. Even by the time they are analyzed, the final number of pages would have increased by many thousands, since thousands of pages pour in every minute. Google claims to have indexed over a trillion pages as of July 25, 2008 [2]. As of February 27, 2012, over 139.2 million websites are hosting these web pages [3].

1.1 Nature of the web

The Web is stateless by nature. Stateless protocol is a communication protocol in computing, which treats each request as an independent request, even if two requests are related to each other. Hypertext Transfer Protocol (HTTP) [5] is the best example of the stateless protocol [4]. Web servers are designed to be stateless [28] in nature which uses the HTTP for the data communication. It treats or processes each HTTP request by an independent process, even if two requests are related to each other. These servers do not store any user data during processing the HTTP request. This is in contrast with the traditional File Transfer Protocol (FTP) [6] servers, which are stateful. In traditional client/server applications, a process will be assigned for each client, until the client terminates. All the requests generated by the client are processed by a single process respectively. These servers will store the client's details during processing each request. The main cause for the Web's stateless nature is performance. The web servers needed to

address the large number of clients than traditional client/server applications, so they do not want any process to be tied up with a single client.

Shopping cart is one of the most common web applications, which requires the application to keep track of the items in the cart during the traverses from one page to another. This makes most of the web applications to be stateful. The web servers support these applications by using session concept. For example, for the Amazon shopping cart application, when the web server notices a new user browsing, it assigns a session ID to the user which will be stored in the browser cookie. This cookie will be sent along with each HTTP request so that the server indentifies the request session. This is how a stateless web server will support a stateful application.

Building stateful applications on the stateless web infrastructure has raised many security problems [7]. Furthermore complicating matters, the Web continues to evolve with new browser features, protocols and standards added at rapid pace. The specifications of new features are often complex, lack of security models and the security of the applications is overlooked. As a result, large numbers of vulnerabilities and security threats are raised for the web applications.

1.2 Contents of the web

Web had been designed for serving static contents; initially, this originated from a single trusted source. It has now evolved into quite dynamic contents and requests derived from multiple sources with varying levels of trustworthiness. Contents may be included by the Web itself, derived from user supplied text or from partially third parties.

Web contents are divided into three types based on the varying levels of trustworthiness [7].

1.2.1 Trusted contents

The contents which are originated from the web application itself are considered as trusted contents. Trustworthiness of these contents depends on the nature of the application and the procedure followed by the application developer. For example, to update status, to write on a wall, to ask a question, to add photos or videos and confirm friend requests on Facebook, all of these are the trusted contents which are generated and maintained by the Facebook application.

1.2.2 Untrusted contents

Many web applications now include the user provided contents such as blogs, comments, feedback, user profiles, etc., in their pages. These contents are the third-party data and less trustworthy than the first-party contents generated by the web application itself. For example, untrusted contents include advertisements and fake profiles in social networks like Facebook and Orkut. The current web, due to its stateless nature, cannot restrict in assigning access privileges to the contents based on their trustworthiness.

1.2.3 Partially trusted contents

Many web applications allow extensions to their pages i.e., they include the links to third-party programs or directly include third-party programs in their pages and run those programs in the browser. For example, third-party applications like CastleVille are embedded in a user's Facebook page, which will collect information from a user's

account and run on the third-party application servers. These contents can be dangerous if they are vulnerable or malicious.

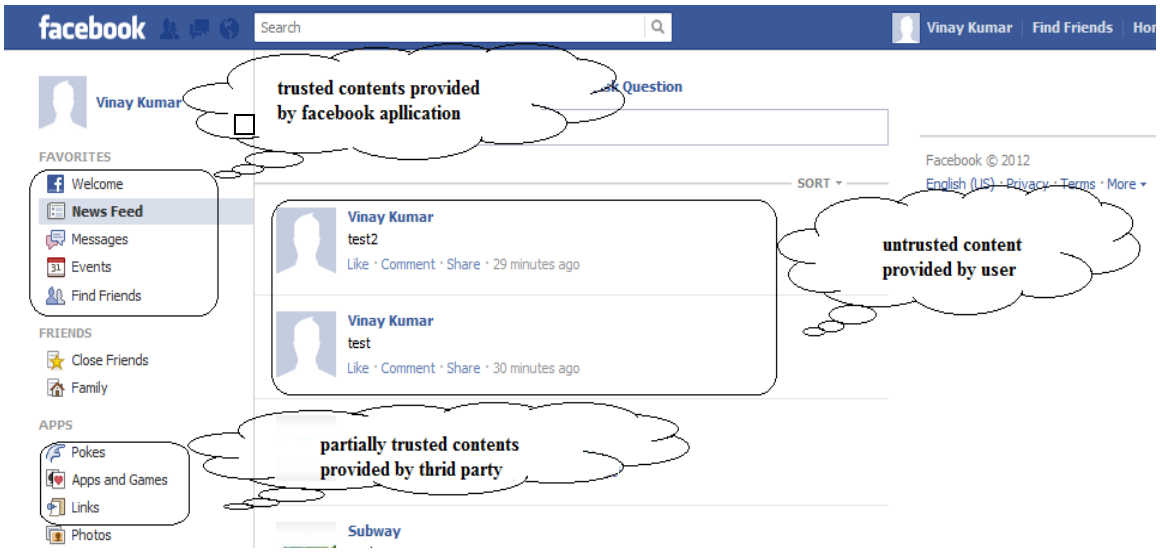

Figure 1: Example for different types of web contents

Figure 1 shows the Facebook application with different types of web contents. If these contents are not carefully handled, malicious code can be injected into the web application. However, Facebook has its own mechanism to handle these contents without any security issues.

1.3 Attacks on web

The Web, due to its importance and stateless nature has become a preferred target of attacks. Web attacks are apparently more serious when they are inflicted upon businesses that store sensitive data, such as personal, military, confidential, medical, governmental, and financial records. The consequences of attacks on any entity range from mildly inconvenient to completely debilitating. According to the Norton

Cybercrime Report 2011 [8], the total cost of cybercrime is at $388 billion per year, which includes $114 billion in direct theft and time spent resolving attacks, plus another $274 billion for productive time victims lost due to cybercrimes being committed against them.

The top ten web attacks according to the OWSAP Top 10, 2007 [9] are

1. Cross Site Scripting (XSS).
2. Injection Flaws.
3. Malicious File Execution.
4. Insecure Direct Object Reference.
5. Cross Site Request Forgery (CSRF).
6. Information Leakage and Improper Error Handling.
7. Broken Authentication and Session Management.
8. Insecure Cryptographic Storage.
9. Insecure Communications.
10. Failure to Restrict URL Access.

Most of the Web vulnerabilities appear to be caused by the mistakes made during the design and development of the application by the developer. However, when we take a deep look at the architecture and functionalities of the Web, we come to the conclusion that the main cause for web vulnerabilities is the access control system of the Web, not the developer.

1.3.1 Access control system

Access control system refers to a security enforcement model that has the ability to decide who can do what to whom in a system. Access control system consists of three components: principals, objects and the access control model. Principals (who) are the entities in the system that can manipulate resources. Objects (whom) are the resources in the system that require controlled access. The access control model tells how decisions are made in the system. For example, consider an online exam application for school. Alice is a professor who teaches CS600 and wants to conduct an online exam for students. To avoid plagiarism, she designed an exam pattern in such a way that each student will get his or her own exam paper based on their ID numbers i.e., Bob with ID number 1 will get paper set 1 and John with ID number 21 gets paper set 21. Here the access control system comes into play, which decides who can do what to whom in a system. When the students login to the application with their ID numbers, the access control system first checks their ID numbers, and then assigns the exam paper to each student respectively.

1.3.2 Web components

The current web consists of two major components, the browser and the server, where the effective access control system needs to be implemented. In terms of access control system, the current web has adopted the inadequate same origin policy (SOP) and same session policy (SSP) for the browser and server respectively. This was sufficient for the earlier day's web, which became inadequate to address the protection needs of today's web. Web applications that embed third party content in their web pages cannot

restrict the permissions of the third party code due to the failures of the access control system. In order to overcome this fundamental problem, we have developed an enhanced browser based access control system by enabling dynamic scoping. The objective of our work is to make the access control system address the current web content problems, which will allow the client and trusted web application contents to share the common library and protect web contents from each other, while they still get executed at different trust levels.

CHAPTER 2: THE PURPOSE FOR PROTECTION ENHANCEMENT

This chapter describes the drawbacks of the current access control system and need of its enhancement. The access control system has been implemented on web components, browsers and servers, by adopting the same origin policy (SOP) and the same session policy (SSP) respectively.

2.1 Same origin policy

The same origin policy (SOP) is also called single origin policy. SOP prevents documents or scripts loaded from one origin from getting or setting properties of a document from a different origin. It also allows scripts running on pages originating from the same site to access each other's methods and properties with no specific restrictions [10]. The term "origin" is defined as a combination of the domain name, protocol and port number of the HTML document. Two documents or scripts are considered to be of the same origin if and only if all these values are exactly the same. For instance, http://www.abc.com/jobs.html and http://www.abc.com/price.html belong to the same origin, but http://www.xyz.com/jobs.html and http://www.abc.com/jobs.html don't belong to the same origin as they had different domains. Similarly, http://www.abc.com/jobs.html and https://www.abc.com/price.html don't belong to the same origin as they had different protocols.

The following example will illustrate the importance of the same origin policy of a browser. Assume that you are logged into Facebook and visit a malicious website in another browser tab. Without the same origin policy, JavaScript on a malicious website

could do anything to your Facebook account. For example, the hacker could read private messages, post status updates, and change security questions. In Figure 2, www.abc.com can access the contents of the www.facebook.com the user page.

In order to avoid this illegal access to Facebook, it is important for the browser to detect trusted and untrusted Java Scripts to access Facebook resources. That's where the same origin policy comes into play. If the JavaScript is included in Facebook HTML page, it may access facebook.com resources; otherwise it cannot access facebook.com resources.

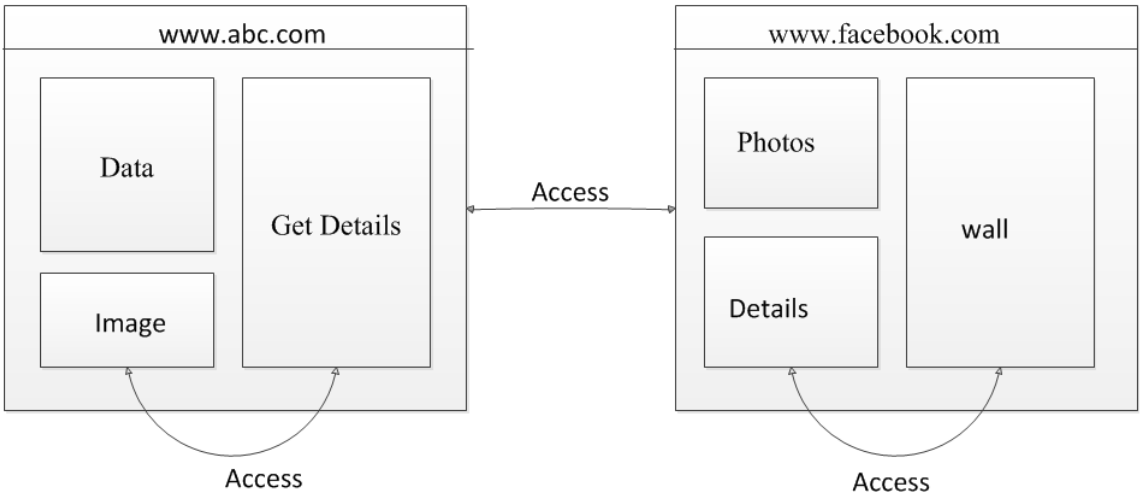

Figure 2: Without same origin policy

In Figure 3 www.abc.com cannot access the contents of the www.facebook.com user page due to the same origin policy. Privileges should be assigned to contents based on the trustworthiness even if they belong to the same origin and this is indispensable in the current web. Cross Site Scripting (XXS) [11] is one of the side effects of the same origin policy

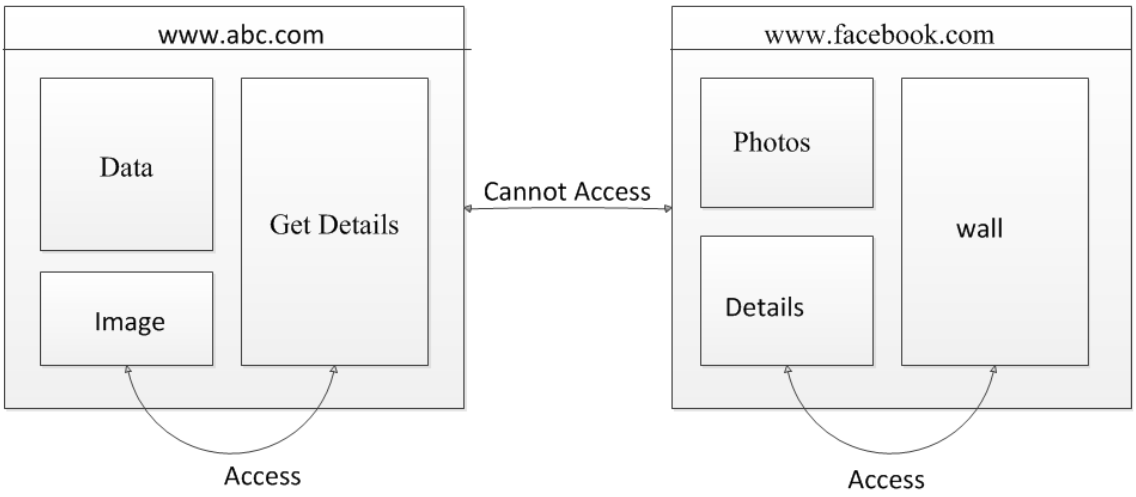

Figure 3: With same origin policy

2.1.1 Cross Site Scripting (XXS)

Cross Site Scripting attack is an injection type of attack that takes advantage of website vulnerability in which the site displays content that includes un-sanitized user-provided data. XSS allows the user to inject a malicious code into trusted websites, which provides attackers a way to bypass client-side security mechanisms (i.e., same origin policy) normally imposed on the web content by modern web browsers. On the successful injection of the code, the attacker can gain elevated access privileges to the entire page based on the same origin policy, i.e., scripts running on pages originating from the same site are allowed to access each other's methods and properties without considering trustworthiness of contents.

For instance, a victim website which allows users to create communities with their own rules, ranks, chat boards and polls. These communities may be designed with

images, graphics, animations and text to make their community look better and more fit the theme. For example, a community that protests against a war might be designed with pictures of recent wars and their consequences. The attacker can inject a malicious code in to the victim website while creating the user communities. As a malicious code originating from the same site, it has access to other scripts or contents in the page based on the same origin policy.

On successful injection of a malicious code and browsing of attackers communities by users, the attacker can take control of user accounts and either use a malicious code to automatically manipulate the user accounts, such as forcing the user to post comments or join the community whether they want to or not, or stealing the credit card and private information. This could also be used to redirect the user to websites that places virii, spyware, adware, or other malicious content on computer.

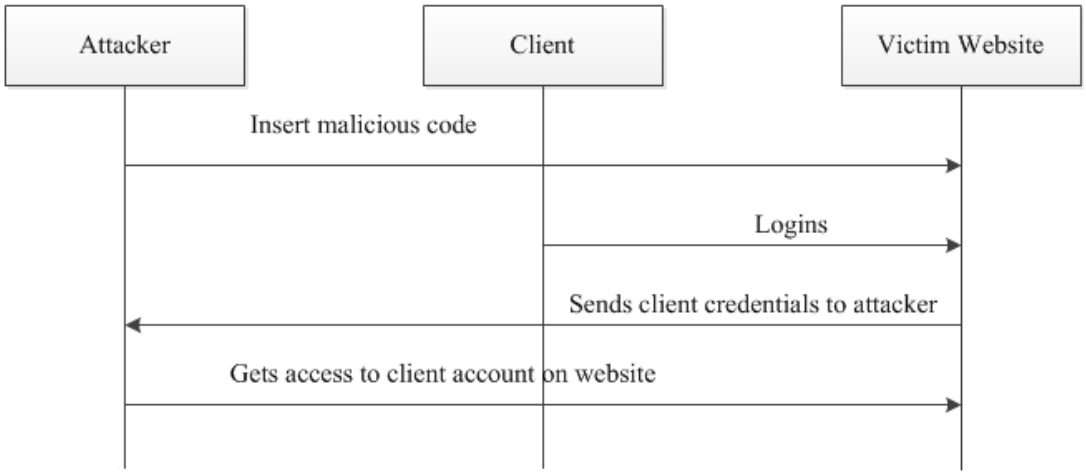

Figure 4: Illustrate XSS Attack

The cause of this attack is due to the inadequacy of current same origin policy which cannot provide the security based on the trustworthiness of current web page contents. Figure 4 will illustrate the sequence of actions performed by XSS attack.

2.2 Same session policy

Similarly, on the server side access control is primarily based on the same session policy. When a user logs into a web application, the server creates a dedicated session for this user, separating him or 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. In the current access control system, it is difficult to allow the request from the same web page to access the same session, while preventing some of them from invoking certain server-side services [12]**.** Cross Site Request Forgery (CSRF) [13] is one of the side effects of the same session policy.

2.2.1 Cross Site Request Forgery (CSRF)

Cross Site Request Forgery is also known as the one-click attack, sea surf attack or confused deputy attack. CSRF is a type of attack on a website in which an intruder masquerades as a legitimate and trusted user. A CSRF attack can be executed by stealing the identity of an existing user and then hacking into a web server using that identity. An attacker can masquerade as a legitimate user by sending HTTP requests that return sensitive user data to the intruder. CSRF exploits the trust that a site has in a user's

browser where as XSS exploits the trust a user has for a particular site [14]. CSRF uses the vulnerabilities in same session policy to perform an attack successfully, i.e., requests or actions which are originating from the same session will be given the same privileges regardless of whether they are originated from first party or third party contents.

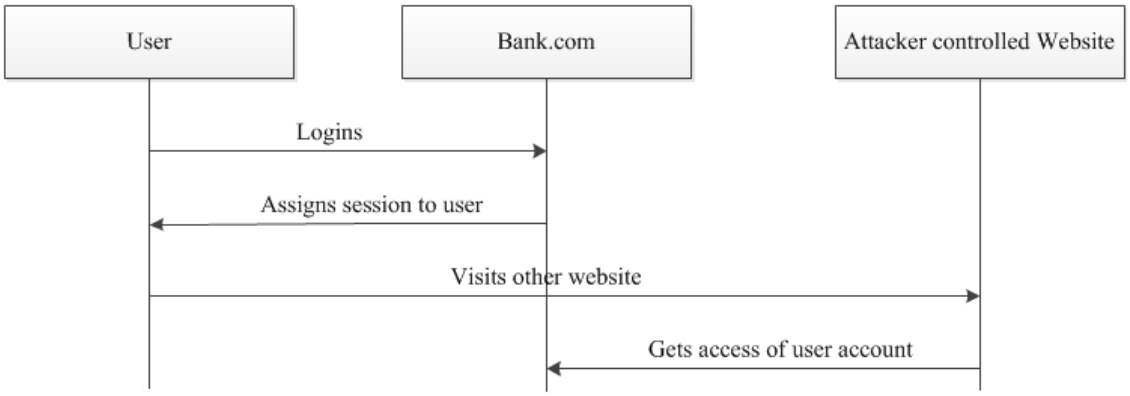

Figure 5: Illustrate CSRF Attack

A real world example of this is the ability of attackers to commandeer certain banking websites. The required steps to gain ownership and perform fraudulent financial transaction are shown in the following example:

1. Once a user logs into an online banking account, the banking server assigns a session to the user.
2. Before the session expires or the user logs off from the banking account, he or she surfs the internet by opening a new tab in the browser.
3. The website surfed by the user contains a hidden code. User browsing activates the code and sends a HTML request to the bank web server with authentication details from browser cookies.

4. So the attacker can make fraudulent transaction to his or her account.

This attack will be successful only when the request is made from a session. Figure 5 will illustrate the sequence of actions performed by a CSRF attack.

2.3 Failure to support design principles

Both the same origin policy and the same session policy failed to fulfill the fundamentals proposed by Saltzer and Schroeder [15] in “The Protection of Information in Computer Systems”. Separation of privilege and least privilege are the two of eight design principles summarized by Saltzer and Schroeder, which are violated by the current web access control system polices. In order to provide efficient security on the Web, the following two principles must be supported by current web access control system.

2.3.1 Separation of privilege

According to the principle of a separation of privilege, privileges in a system should be divided into less powerful privileges, such that no single accident or breach of trust is sufficient to compromise the protected information. For instance, this principle is most commonly used in the banking system for bank safety deposit boxes, where two physical keys are needed to lock and unlock the boxes. Once the box is locked, two keys are separated and maintained, one by a user and another by the bank manager to avoid unauthorized access due to loss of keys.

2.3.2 Least privilege

According to the principle of least privilege, each user in the system should be least privileged to complete their jobs without any interruption based on their trustworthiness. For instance, in UNIX the normal user should not be given the privileges of a root user unless they are required for its legitimate purpose.

The current web access control systems are inadequate to address the protection needs of today's web because it is violating the above mentioned design principles. So there is need for redesigning the access control system of the Web to provide efficient security. We have enhanced the access control system of the Web by enabling the dynamic scoping for the browser, which overcomes the inadequacies in the same origin and the same session policies and also provides support for the Saltzer and Schroeder design principles.

CHAPTER 3: RELATED WORK

The need for enhancing the fine-grained access control system for the Web has been recognized earlier by many researchers. A number of approaches are proposed by researchers in two ways: either to modify the browser or rewrite the entire script, which can be done either statically or dynamically.

By using the iframe [16], we can easily isolate the third-party contents or script by putting them in the iframe from the host page. Scripts included in the iframe will be considered as originating from the different origin, so those scripts cannot access any script or contents in the host domain. This will have a severe effect on the web application's functionality. To avoid this all-or-nothing model, several solutions were proposed for a browser-based access control system.

Crites et al.'s proposed Mashup solutions [24]. Mashup solutions brought a policy that abandons the same origin policy by allowing the integrator to specify public and private data including DOM access. Completely abandoning the SOP would require a significant change to websites. This is going to be expensive work.

To avoid completely abandoning the SOP, Miller proposed a Caja method [25]. This approach is based on a concept of rewriting the program source code to enforce the security policies. The rewriting procedure of Caja is very complicated and cannot always preserve original script functionality.

In contrast to Caja, Barth et al.'s isolated world mechanism [23] replaces the one-to-one context mapping with a one-to-many map where each context maintains a

mapping table to the DOM elements of the host page. This ensures that only host objects are shared among all worlds but not native or custom objects. We have adopted this isolated world mechanism idea to isolate the contents' execution.

Zhou and Evans proposed a solution [26] in extension to isolated world mechanism. It is a one-way trust model with a goal to protect user content from untrusted scripts rather than to protect embedded scripts from the host page or each other. This approach doesn't consider the JavaScript frameworks like jQuery and other attacks like cross site scripting, which are very important to provide the security to web applications. This fine-grained access control system aims to protect the trusted content from the untrusted content, but not to protect the contents from each other.

Du et al.'s proposed SCUTA [12]. It is based on the ESCUDO [27], which was their earlier work in protecting privacy for web applications. SCUTA uses the new concept called sub-session for web applications, which is based on the ring concept in the ESCUDO, so the requests from trusted client-side contents can be separated from those of untrusted contents; such a separation enables web applications to enforce a fine-grained access control system. This approach provides security measures against various attacks like cross site scripting, which are not addressed in the Zhou and Evans approach.

In both solutions, the JavaScript code in different worlds or rings will not interact with each other. In a real world application, especially in many social networking sites, it would be ideal that the hosting applications have the capability to provide a shared library, which can be used by third party users. Based on this observation, we propose to

use the origin of the function call, instead of the location of the function, to decide the privileges of the JavaScript code.

CHAPTER 4:TWO-WAY SECURITY MODEL

The objective of our work is to make the access control system address the current web content problems. The proposed approach will allow the client and trusted web application contents to share the common library and protect web contents from each other by executing at different trust levels. We assume a two-way security model since our goal is to protect web contents from each other and allow sharing the common library among the web contents.

We need to make fundamental changes to the current web protection model to address the protection needs of modern applications. The two-way security model can be obtained by enabling dynamic scoping for the current web access control system. The two-way security model doesn't target in changing the today's web architecture but focuses on fundamental changes to the access control system.

Our model doesn't make any changes to the basic policies of current access control system but enhances it with dynamic scoping, i.e., our model will use the existing same origin and same session policies without any changes. Our model allows the developer to configure their application by appropriately specifying the shared library and other contents with their trustworthiness. Web applications communicate the configuration to the web browser, where the proposed access control model enforces access decisions based on the configuration. Figure 6 will illustrate our two-way security model.

1. Let us consider Group 1 and Group 2 are the client and trusted contents, and the shared library is a collection of some trusted contents.
2. The application developer specifies the shared library and other contents with their trustworthiness to the browser.
3. Group 1 and Group 2 can access (read-only access) shared library but cannot manipulate it, i.e., Group 1 and Group 2 can get the resources from the shared library and use them, but cannot make any changes to the shared library. This proposal is based on a very simple principle: If one would like to manipulate his own work, it is allowed to proceed; if one would like to manipulate something outside his work, the actions will be prohibited.
4. Group 1 and Group 2 cannot access each other.

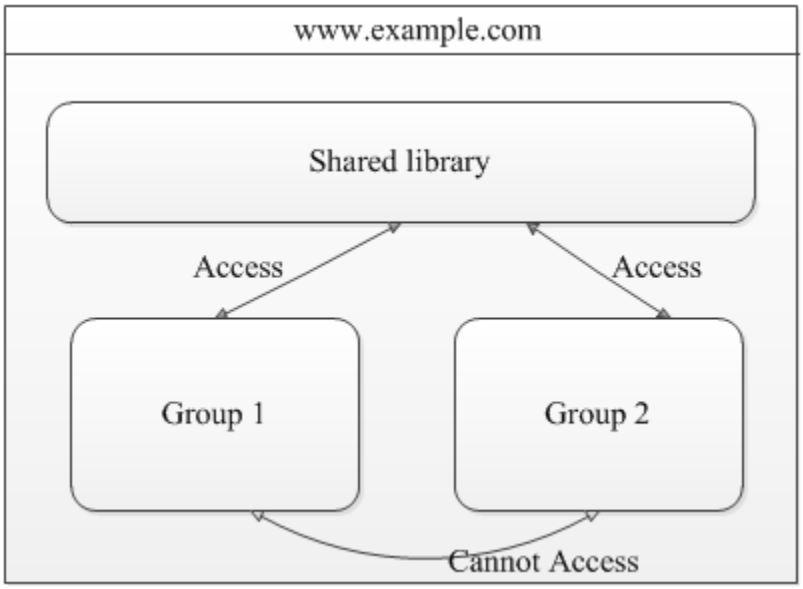

Figure 6: Two-way security model

In modern web applications, different web contents are needed to access each other to perform their task successfully. The complete isolation of contents will not fit for modern web applications, so there is a need for the web applications to share some common things among the web contents irrespective of trustworthiness. To avoid the burden and complexities in defining relationships among the web contents and protecting them from each other, our model defines the shared library and allows web contents to access them. To differentiate and designate the trustworthiness of different components in a web application, we introduce the group concept for different components in a web page. This concept is very similar to the ring concept in SCUTA. The key difference is that the defining access control is based on where the JavaScript code is initiated, not where the code is located.

Our two-way security model places all the web contents in different groups based on their trustworthiness except the shared library. Shared library contents are placed in the default group. Our model allows web developers to choose the total number of groups that fit their application needs. The number of groups for one application is independent from others.

This two-way security model can be achieved by enforcing dynamic scoping for web based access control system. In computer programming, scope is the range within a computer program in which a variable name or other identifier is valid and can be used, or within which a declaration has effect. Computer programming has two different types of scoping: they are static and dynamic scoping.

Static scope is determined at compile time by the compiler using a sequential processing of program and remains the same throughout the program. Static scoping determines the occurrence of an identifier by first checking the local block in which the name appears, then the block construct that declares the block (i.e., its static parent). This process is repeated until a definition is found. That is, the compiler first searches (searching for variable or identifier) in the local function (the function which is running now), then searches in the function in which that function was defined, then searches in the function in which that function was defined, and so forth until a definition is found. By default C, C++ and JavaScript uses static scoping.

In contrast, dynamic scoping is determined at runtime. In dynamic scoping, processing of program statements follows the execution order of different statements and can change during the execution of the program. Dynamic scoping determines the definition for an occurrence of the identifier or a variable by examining the calling sequence, rather than the program block declaration hierarchy as in static scoping. That is, the search for identifier starts first in local function, then search in the function that called the local function, then search in the function that called that function, and so on, up the call stack until the definition is found. "Dynamic" refers to change, in that the call stack can be different every time a given function is called, and so the function might hit different variables depending on where it is called from. Figure 7 will illustrate the difference between the static and dynamic scoping.

Static scoping	Dynamic scoping

int a =10;
int Bob()
{
int b= a+5;
return b;
}
int Alice()
{
int a=20;
return Bob();
}
int main()
{
Bob();//returns 15
Alice(); //returns 15
return 0;
}
	int a =10;
	int Bob()
	{
	int b= a+5;
	return b;
	}
	int Alice()
	{
	int a=20;
	return Bob();
	}
	int main()
	{
	Bob();//returns 15
	Alice(); //returns 25
	return 0;
	}

Figure 7: Difference between static and dynamic scoping.

1. Bob's function returns the 15 by fetching value from its lexically enclosing scope i.e. a=10, when it is called directly. The Function Alice() which calls Bob(), returns different values in static and dynamic scoping.
2. In static scoping the function, Alice () calls Bob (), which fetches the variable "a" value from its lexically enclosing scope i.e., a=10 and returns 15.
3. In dynamic scoping the function, Alice () calls Bob (), which fetches the variable "a" value from the initiated function i.e., a=20 and returns 25.

Execution of JavaScript requires a scope for top-level script variable storage as well as a place to find standard objects like function and object. Calls to functions in JavaScript use static scope, which means that variables are first looked up in the function and then, if not found there, in the lexically enclosing scope. This causes problems if

functions you define in your shared library need access to variables you define in your instance scope as illustrated in Figure 7. For better understanding of how scoping affects functionalities of shared library consider Bob() as shared library function and Alice() as initiated function in Figure 7. Therefore our two-way security model can be achieved only by enforcing dynamic scoping for web based access control system.

In order to get better understanding of our two-way security model, we use the more complete example shown in Figure **8** to demonstrate working of our model. In this example, shared library contains variables a=6, b=9, and functions product(), and reset(). The product() function calculates and returns the product of two numbers. The reset() function will manipulate the contents of documents such as making the document empty or setting different values to the variables. The remaining scripts are grouped into group1 and group2 according to their trustworthiness.

Group1 contains the variables a=1, b=2, and a call to the product() function in the shared library. The product() function initiated from group1 will fetch values of the variables a and b from the group1, and returns 2 rather than 54.

Group2 contains the variables a=1, b=2, and a call to the reset() and product() functions in the shared library. The product() function initiated from group2 will fetch values of the variables a and b from the group1 and returns 2 rather than 54. When the dynamic scoping is used, DOM root is the root of scripts scope that initiates the function rather than the scripts which contain the function. The reset() function initiated from group2, will manipulate the contents of group2 only, not the shared library as the document root is the root of group2 rather than the root of the shared library.

```
<html>
<head>
<script type="text/javascript">
        var a=6,b=9;
        function product()
        {
                return a*b;
        }
        function reset()
        {
                //method manipulates contents of document
                  document.write("document made empty");
        }
</script>
</head>
<body>
<script type="text/javascript" Group="1">
        var a=1,b=2;
        product();
</script>
<script type="text/javascript" Group="2">
        var a=1,b=2;
        reset();
        product();
</script>
</body>
</html>
```

Figure 8: Complete example demonstrates working of a two-way security model

CHAPTER 5: EFFECTUATION OF TWO-WAY SECURITY MODEL

This chapter describes a prototype implementation of the two-way security model on the Lobo browser [17] based on the requirements and design presented in Chapter 4. The Lobo open source project aims to develop an extensible browser and RIA platform written completely in Java that not only supports HTML and JavaScript, but also enables rendering of arbitrary Rich Internet Application (RIA) languages [18]. The Lobo browser is built on the Cobra HTML Rendering engine, which is a pure Java HTML renderer and DOM parser that is being developed to support HTML 4, JavaScript and CSS 2. Cobra uses the Rhino 1.6R5 JavaScript engine, which is released by the Mozilla Foundation [19].

5.1 Lobo Architecture

The architecture of the Lobo browser which we derived is shown in Figure 9 [20]. Lobo is intended to be a platform for building new client-side web languages. Therefore, the browser architecture is designed to be easily extensible. It comprises five major subsystems plus the dependencies between them.

5.1.1 User Interface

The User Interface subsystem is the layer between the user and the browser engine. It provides features such as toolbars, page services, navigation, preferences, and printing. It may be integrated with the desktop environment to provide browser session management or communication with other desktop applications.

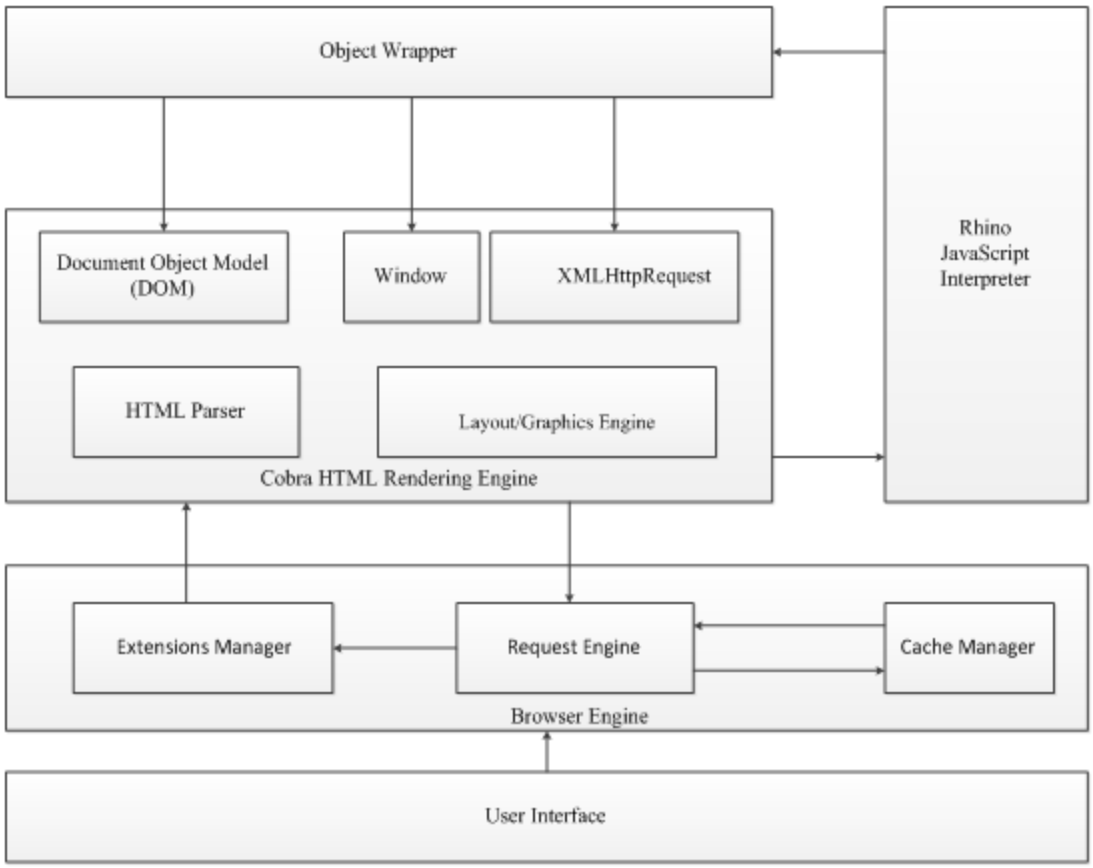

Figure 9: Architecture of Lobo browser

5.1.2 Browser Engine

The Browser Engine subsystem is an embeddable component that provides a high-level interface to the rendering engine. It consists of three important components: the Request Engine, Extensions Manager, and Cache Manager. User requests are forwarded to the servers by the Request Engine from the user interface. The Request Engine uses the Extensions Manager to choose an appropriate extension to render the response. The Extensions Manager uses the Cobra HTML Rendering Engine for rendering web pages. The Cache Manager is responsible for caching responses based on the instructions specified in the HTTP cache-control header. The Request Engine interacts or contacts with the

Cache Manager before issuing a network request, and serves the response from the cache if possible.

5.1.3 Cobra HTML Rendering Engine

The Cobra HTML Rendering Engine subsystem is a pure Java HTML renderer and DOM parser that produces a visual representation for a given URL. It is capable of displaying HTML and Extensible Markup Language documents, styled with CSS, as well as embedded content such as images. It consists of five major components, which are HTML parser, Layout or Graphics Engine, Document Object Model (DOM), Window and XMLHttpRequest objects. Cobra uses the HTML parser, which can be used independently of the Cobra HTML Rendering Engine to parse the web page to page and construct a DOM tree corresponding to the page. Each web page is assigned a distinct DOM and a Window, which is an abstraction of the window in which the web page is displayed. The XMLHttpRequest object is used by JavaScript programs to send HTTP requests. The Layout or Graphics Engine is used to render the graphic contents of web pages.

5.1.4 Rhino JavaScript Interpreter

Cobra uses the Rhino 1.6R5 JavaScript engine, which is released by the Mozilla Foundation [19]. Rhino JavaScript interpreter executes JavaScript code, which may be embedded in web pages. Rhino doesn’t contain any objects or methods for manipulating HTML documents but it is only an implementation of core language [21].

Rhino includes the following features [21]

1. All the features of JavaScript 1.5
2. Allows direct scripting of Java
3. A JavaScript shell for executing JavaScript scripts
4. A JavaScript compiler to transform JavaScript source files into Java class files.

5.1.5 Object Wrapper

The Rhino JavaScript Interpreter accesses the DOM, Window, and XMLHttpRequest objects via the Object Wrapper. All the requests to the three objects are mediated through Object Wrapper.

5.2 Identifying subsystems of the Lobo browser architecture for implementation

This section describes the identification of the subsystems of Lobo browser architecture to make modification for enforcement of two-way security model. JavaScript is a dynamic scripting language, which is one of the sources for the attackers to violate the security policies of web page. In Lobo browser, JavaScript is parsed by HTML parser and executed by the Rhino JavaScript engine. Rhino was completely written in Java and enforces its own security policies. It is very important to understand the terms context and scopes in Rhino.

The Rhino context object is used to store thread-specific information about the execution environment [22]. A thread executing JavaScript should be associated with only one context.

Execution of JavaScript requires a scope to find a place where it can access and store the variables or objects. In Rhino it is important to understand that scope is independent of context that created it, i.e., creating a scope for JavaScript can be done using one context and executing the script using that scope and different context is allowed.

Rhino follows the same origin policy, which assigns the privileges based on the origin. Rhino provides the ability to keep track of the origin of a code in webpage. Rhino provides a security-channel to enforce its security features in web application. The security channel needs to do two things.

First, every context that is created must be supplied an instance of an object that implements the SecuritySupport interface. This will provide Rhino the support functionality it needs to perform security-related tasks [21].

Second, the value of the property security.requireSecurityDomain should be changed to true in the resource bundle org.mozila.javascript.resources.Security. The value of this property can be determined at runtime by calling the isSecurityDomainRequired method of context. Setting this property to true requires that any calls that compile or evaluate JavaScript must supply a security domain object of any object type that will be used to identify JavaScript code [21].

The security-channel provided by Rhino will be sufficient for overcoming the current web access control drawbacks; by implementing our two-way security model without any modifications to the current security policy of Rhino. We need to make modification to the Cobra HTML Rendering Engine subsystem rather than the Rhino engine security features to implement the two-way security model.

5.3 Two-way security model implementation

This section describes a prototype implementation of a two-way security model on the Lobo browser. Our two-way security model implementation was involved in adding or modifying approximately 900 lines of code to the Cobra HTML Rendering Engine. We did not make any modification to the Rhino JavaScript engine security features. Hence, our implementation can be used with any pure Java based web browser that uses the Rhino JavaScript engine. Our implementation involved two phases:

1. Extracting and Tracking security groups.
2. Enforcing access control policy.

5.3.1 Extracting and Tracking security groups

This phase deals with the Extracting and Tracking security groups of two-way security model. Whenever web application or page is called from Lobo web browser, Cobra HTML Rendering Engine parser parses the web page and constructs the DOM objects. We have modified the Lobo browser to recognize a new attribute group in script tags. During this process our two-way security model extracts the security group from script tags and stores it in the DOM elements for the respective HTML tags. If a group

element is not found in the script tags our model assigns a default group to that content. The contents with a default group are categorized as shared contents. It is the responsibility of the developer to configure their application with different security groups.

Two-way security model tracks the security groups during the execution of scripts. It maintains a webpage-specific table, which is used for maintaining security groups of current executing web contents. Our model dynamically updates the table according to the flow of execution. Our model does not make any changes to the order of parsing and execution of the web contents. Normally, the parsing and execution of the web contents will be done in the order of their appearance and dependencies on the web page. The common processing work of the Cobra HTML Rendering Engine parser is creating DOM elements and adding them to the DOM tree. Some web contents can momentarily create HTTP request for accessing other web contents during the processing work of the Cobra HTML Rendering Engine. Before answering those requests, our model retrieves the security group of HTTP requests content origin from the DOM and updates the web-page specific table and then answers the request. As a result the new requests generated dynamically can still execute in the origin context.

5.3.2 Enforcing access control policy

Two-way security model enforces the access control policy based on principle that the contents of web application share the common library and protect from each other, while they still get executed at different trust levels. Two-way security model enforcement comprises three parts.

First, our model isolates all the web contents based on their group values. In order to isolate the web contents we adopted the isolated world mechanism idea from Adam Barth's Protecting Browsers from Extensions Vulnerabilities [23]. The isolated world mechanism replaces the one-to-one context mapping with a one-to-many map where each context maintains mapping table to the DOM elements of the host page. This ensures that only host objects are shared among all worlds, but not native or custom objects.

We adopted and modified this mechanism to implement the two-way security model. Our model creates a separate context for each group. Each time the Rhino JavaScript engine is invoked by the Cobra HTML parse to execute JavaScript program, it passes the JavaScript context corresponding to the programs group. As a result, JavaScript programs belonging to a group can access only the custom and native objects that reside in the context belonging to the group. This isolation is necessary to protect the web contents from each other.

Second, our model supports the dynamic scoping and scripting as we are enforcing access control policies at runtime. As our model creates separate context for each scripts, the dynamically generated scripts will run in different context from the scripts that created them. This will break the functionality since variables and functions that should be shared are now isolated. We made modifications to the Lobo Browser in such a way that dynamically generated scripts will inherit the group from their creator, thus executing within the same context.

Third, our model supports the library sharing by modifying the prototype chain of scope and restricting any modification to the shared library by using sealObject() method.

Our model enables the sharing of contents by creating a new context and calling scope object by setting its setprototype method to sharedscope object and parent to null. Our model restricts others making changes to shared library by calling the sealObject method. sealObject method will not allow to add or delete properties to the object and make changes to the existing objects. Our model assigns the default group to the contents that don’t carry the group element in the script tags. The content with default group is categorized as shared library. It is the responsibility of the developer to specify the type of content by configuring the web applications.

We have implemented two phases of the two-way security model without any compatibility issues.

CHAPTER 6: CONCLUSION

We strongly believe that the access control system in the current web is inadequate to satisfy the protection needs of today's web. The web technology is still evolving, so a good access control system 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. So we outlined the two characteristics that a security model of the access control system should adapt, to address the current web problems and provide support to the security model evolution that address the future web problems. We have presented a browser based access control by enabling the dynamic scoping. This access control model is systematically designed to fulfill the two characteristic requirements using mandatory access-control principles. We implemented a prototype of a new browser based access control in the Lobo web browser and illustrated how web applications can use this new access control system.

Future research in browser access control should consider how to facilitate richer web applications while enforcing the principle of least privilege. In the future, web applications will feature richer and more interactive clients executing in the web browser. So the future research should focus on architecture improvements of the Web and design of API methods, to facilitate JavaScript programs to enforce the least privilege principle of the access control in the richer applications.

BIBLIOGRAPHY

[1] World Internet Usage and Population Statistics. Retrieved from Internet World Stats: http://www.internetworldstats.com/stats.htm

[2] We knew the Web was Big. Retrieved from Google Official Blog: http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html

[3] Domain Counts and Internet Statistics. (2012, March 25). Retrieved from Domain Tools: http://www.domaintools.com/internet-statistics/

[4] Stateless Protocol. Retrieved from: http://en.wikipedia.org/wiki/Stateless_server

[5] Wong. C, HTTP Pocket Reference: Hypertext transfer Protocol (p. 80). O'Reilly Media.

[6] FTP – The File Transfer Protocol. (2006). Retrieved from South River Technologies webdrive: http://www.webdrive.com/docs/geninfo/wpftpbasics.pdf

[7] Wenliang, D., Karthick, J., Tan, X., Tongbo, L., & Chapin, a. S. Position paper: Why are There so Many Vulnerabilities in Web Applications? Dept. of Electrical Engineering & Computer Science, Syracuse University, Syracuse, New York, USA.

[8] Norton Study Calculates Cost of Global cybercrime. Retrieved from http://www.symantec.com/about/news/release/article.jsp?prid=20110907_02

[9] Top 10 2007. Retrieved from OWSAP the Open Web Application Security Project: https://www.owasp.org/index.php/Top_10_2007.

[10] Sullivan, B., & Liu, V. (2001). Web Application Security. McGraw-Hill Professional.

[11] Top 10 2007-Cross site scripting. Retrieved from OWSAP the Open Web Application Security Project: https://www.owasp.org/index.php/Top_10_2007-A1.

[12] Wenliang, D., Karthick, D., Tan, X., & Tongbo, L., (2011). SCUTA: A Server-Side Access Control System for Web Applications.

[13] Burns, J. Cross Site Request Forgery-An Introduction to a Common Web Application Weakness. Information Security Partners, LLC.

[14] Cross-site Request Forgery. Retrieved from http://en.wikipedia.org/wiki/Cross-site_request_forgery

[15] Saltzer, J. H., & Schroeder, M. D. (1975). The Protection of Information in Computer Systems. Proceedings of the IEEE.

[16] WHATWG community. HTML Living Standard iframe Element. Retrieved from whatwg: http://www.whatwg.org/specs/web-apps/current-work/#the-iframe-element.

[17] Lobo: Java Web Browser. Retrieved from lobobrowser: http://lobobrowser.org/java-browser.jsp

[18] Lobo Project. Retrieved from lobobrowser: http://lobobrowser.org/index.jsp

[19] Cobra: Java HTML Renderer & Parser. Retrieved from lobobrowser: http://lobobrowser.org/cobra.jsp

[20] Karthick, J. (2011). Protection Models for Web Application. Syracuse University.

[21] RhinoOverview. Retrieved from Mozila : http://www.mozilla.org/rhino/overview.html

[22] Rhino Scopes and Context. Retrieved from Mozilla: http://www.mozilla.org/rhino/scopes.html

[23] Adam, B., Adrienne, P., Prateek, S., & Aaron, B. (2010). Protecting Browsers from Extension Vulnerabilities. 17th Network and Distributed System Security Symposium.

[24] Steven, C., Francis, H. & Hao, C. (2008). OMash:Enabling Secure Web Mashups via Object Abstractions. 15th ACM Conference on Computer and Communication security.

[25] Mark, S., Mike, S., Ben, L., Ihab, A., & Mike, S. (2007). Caja Safe Active Content in Sanitized JavaScript.

[26] Yuchen, Z., & David, E. (2011). Protecting Private Web Content from Embedded Scripts. *ESORICS*, (pp. 60-79).

[27] Karthick, J., Wenliang, D., Balamurugan, R., & Steve, J. C. (2010). ESCUDO: A Fine-Grained Protection Model for Web Browsers. 30th IEEE International Conference on Distributed Computing Systems.

[28] Cooper, S. B. What is a Stateful Protocol. Retrieved from ehow tech: http://www.ehow.com/facts_7454206_stateful-protocol_.html

[29] Internet. Retrieved from http://en.wikipedia.org/wiki/Internet