Web Application FireWall
Project Report

Team Name: Password
Weiliang Xing
Song Bai
Jialing Wu
Hongjin Zhu

Abstraction:

In this project, we implemented a web application firewall (WAF) in local server by building Apache Module to realize signatures and anomaly detection methods. For testing purposes, we built a blog website with vulnerability.

The WAF could successfully detect and refuse the requests from the blog website for XSS attack, SQL injection, and other basic attacks by detecting the attack through signatures and anomaly detection by monitoring and learning the web traffic while not affecting normal requests. Experiments showed its good properties as designed.

Background:

A web application firewall (WAF) is an appliance, server plugin, or filter that applies a set of rules to an HTTP conversation [1]. The firewall controls input, output and access of an application service. The firewall could monitor, accept, drop or block the web traffic, which goes through to gain access based on designed protection rules. The rules mainly cover common basic attack including but not limited cross-site scripting (XSS) and SQL injection. According to the project description, our work mainly focuses on network-based application firewalls, which operate at the application layer of a protocol stack [2]. This type of the firewall was first described by *Gene Spafford*, *el.* [3]. The key benefit is that it can detect whether an unwanted protocol is being sneaked or being abused in any harmful way. For basic purpose, two practical ways to implement the WAF in this project is using a server-proxy with rewriting capabilities or creating an Apache module. In our project, we will use the second option, which will implement a new Apache module working as WAF.

It is necessary to build a web application firewall because many kinds of attacks could compromise the website by utilizing the vulnerability of the website. Cross-site scripting (XSS) is one type of computer security vulnerabilities, which enables to inject client-side script into web pages to bypass access controls like same-origin policy [4]. The basic idea for the WAF is that WAF will recognize the properties of XSS attack and will not permit output validation, as if it did not recognize the context of the data. SQL injection is another type of code injection techniques by inserting malicious SQL statements for execution. SQL injection is always combined with XSS attack in reality. Another attack tries using robot protocol. Robots are normally used by search engines to categorize and archive web sites [5]. However, malware robots can scan for security vulnerabilities. The WAF will try to block abnormal requests while not affect normal visit through signature recognizing and learning.

To defend XSS attacks, the common way is escaping from XSS attacks. Using escaping the browser could be informed that the data you are sending should be treated as data in text way, which is not executable as the scripting intends to. The reason for that is because the XSS attackers always use HTML reserved characters to do cross scripting. To defend against that, escape character using *&#* sequence followed by its character code [7]. For example, replace *#* with *&#35*. Normally, it is not recommended to do escaping by programmer himself, instead using build-in

functions is a better way. In this project, we will not use such way but utilizing signature and anomaly detection to test the correctness and efficiency of the methods we implemented. To defend SQL injection, the normal ways include constraining and sanitizing the input data, using prepared statements instead dynamic SQL, etc. In this project, to ensure the complete vulnerability of the test website, we will not implement any methods for validation. Instead, the building of WAF should ensure the safety of the test website as it aims to.

Goal:

Design, implement and validate a Web Application Firewall and its test website using two methods:

1) Signature, which matches malicious requests, as we want to match for blocking purpose;

2) Anomaly Detection, which recognizes attacks in testing phase by "learning" the properties of normal requests in training phase when passively monitoring the website traffic.

Environment Setting:

-OS and Software

We ran the program in *Linux Ubuntu14.04.2 LTS* with *Apache2.4.7* as local server, *PHP5.5.9* as server-side language and *MySQL5.5.41* as database.

-Database Setting

We create a database *waf_db* in *MySql*, and use user *root* to connect to the database, with the *username=root, and password=123*, you can find tables' details in Data Management Design part.

-Configuration Files

The configuration files are located *in /home/pw/NS/pwd_waf* folder (Change permission to allow Apache read/write) with details as below:

1. *WAF_Admin*: the administrator username and password are required to change between Training and Detection mode. Data Format is : first line "*user=admin*", second line "*password=123*". You should change to desired username/password without any space.
2. *WAF_Sig_Conf*: Signature file which contains characteristic of maliciousness. Data Format is: "*REQUEST_METHOD:GET,PARAMETER:*,CONTAINS:"<script>""*. There is only one line for each rule. No space except comment part or no empty line will be in the end.
3. *WAF_Profile*: The file with generated information for anomaly detection. The user should not modify this file.
4. *WAF_Mode*: The file to store the current mode. The user should not modify this file.

-Running Commands

1. You can directly run the following command in the *folder /home/pw/NS/pwd_waf* folder to generate an Apache Module:

#:sudo apxs -cia -L/usr/lib/mysql -I/usr/include/mysql -lmysqlclient -lm -lz mod_pwd_waf.c

2. And then run this command to restart apache service:

#:sudo service apache2 restart

Data Management Design:

In the whole system, we built one configuration file and two databases for the WAF project. They are stored as *WAF_Sig_Conf, training_data.sql,* and *web_db.sql*.

1. Signature Configuration File (*WAF_Sig_Conf*): It works for signature detections, which stores malicious signatures acting as blacklist rules (Appendix. A).
2. Anomaly Detection Database (*training_data.sql*): It works for "learning" anomaly detection, which can learn from training model. When WAF receives requests from user, it stores the URI and URI's parameters' information (i.e. number, length, parameter name) in the database (Figure1). The Pages table represents the URI and its max number of parameters. In Records table, ID represents the primary key identifier of each URI. The URI's parameter and the length of parameter's value will be correspondingly stored. In Parameters table, more detailed information of URI with the specific parameters will be stored, such as the averages length (average), standard deviation (SD), charset of parameter's value (charset), and count times of its occurrence (count). Note that table Parameters is relational to table Records, which linked to every recorded URI in WAF's anomaly detection database. (Note: URI, or Uniform resource identifier, is a string of characters used to identify a name of a resource [6]. For example, the URL *http://localhost/project2/index.php?id=1* refers to a resource identified as */project2/index.php*)

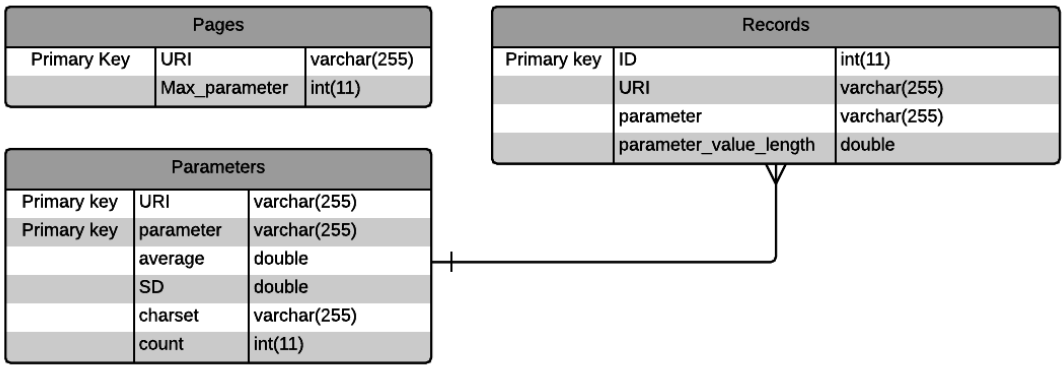

Figure1. Anomaly Detection Database.

3. Web Application Database (*web_db.sql*): It works for our web application - a blog website to manage users, blogs and blog comments. There are three relational tables stores basic information for users, user's blogs and blog's comments in table users, blogs, and comments (Figure 2). In table users, the table stores every user's basic information including

username, password, gender, major, annual_salary, remark and time stamp for every entity; table blogs stores detailed information of blogs for each user in user table, including title, view counts, content, and time stamp of that entity, with many-a relation to table users; similarly, table comments has many-a relation to table blogs, with each blog's comments' detailed information including its author, content and

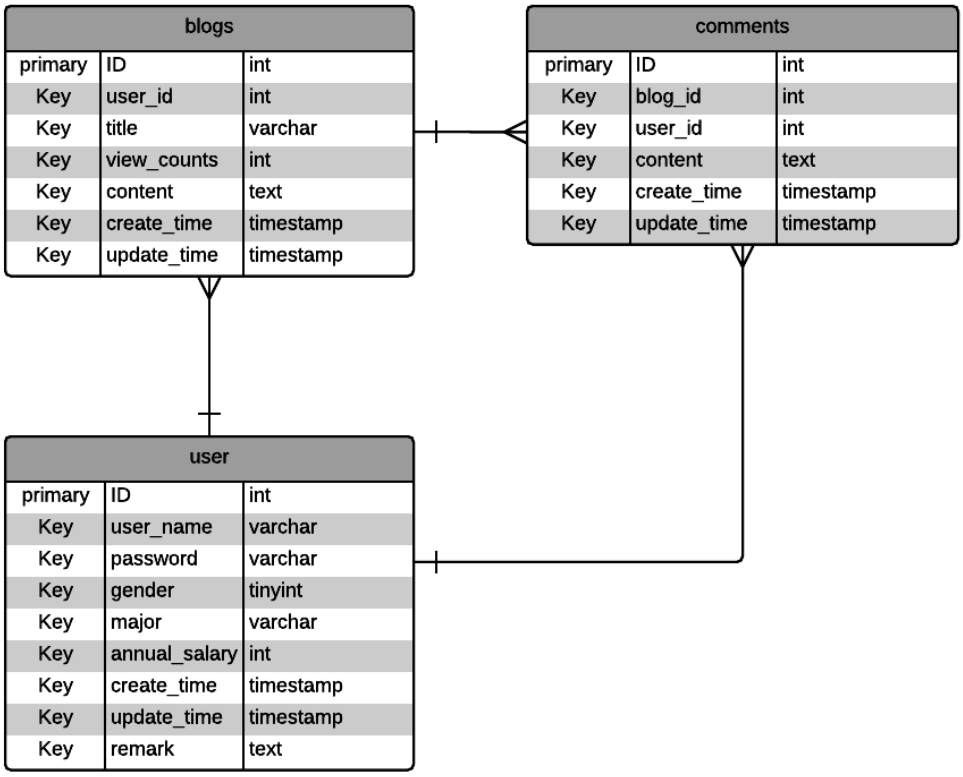

Figure 2. Web Application Database

Module Design:

There are two types of methods implemented in our Apache modules for WAF. One is signature detection, and the other is anomaly detection.

1. Signature detection. The total detection procedure based on signature method is described as flow chart shows (Figure3). The basic logic is: in Apache module, we build rules stored in database; when request from client comes, it will be filtered in such module. The filtering will decide whether the request is malicious or not by matching its content with the rules. The rules act as blacklist, which means it will reject the request when matching with any rule, while allows accessing to web application otherwise.

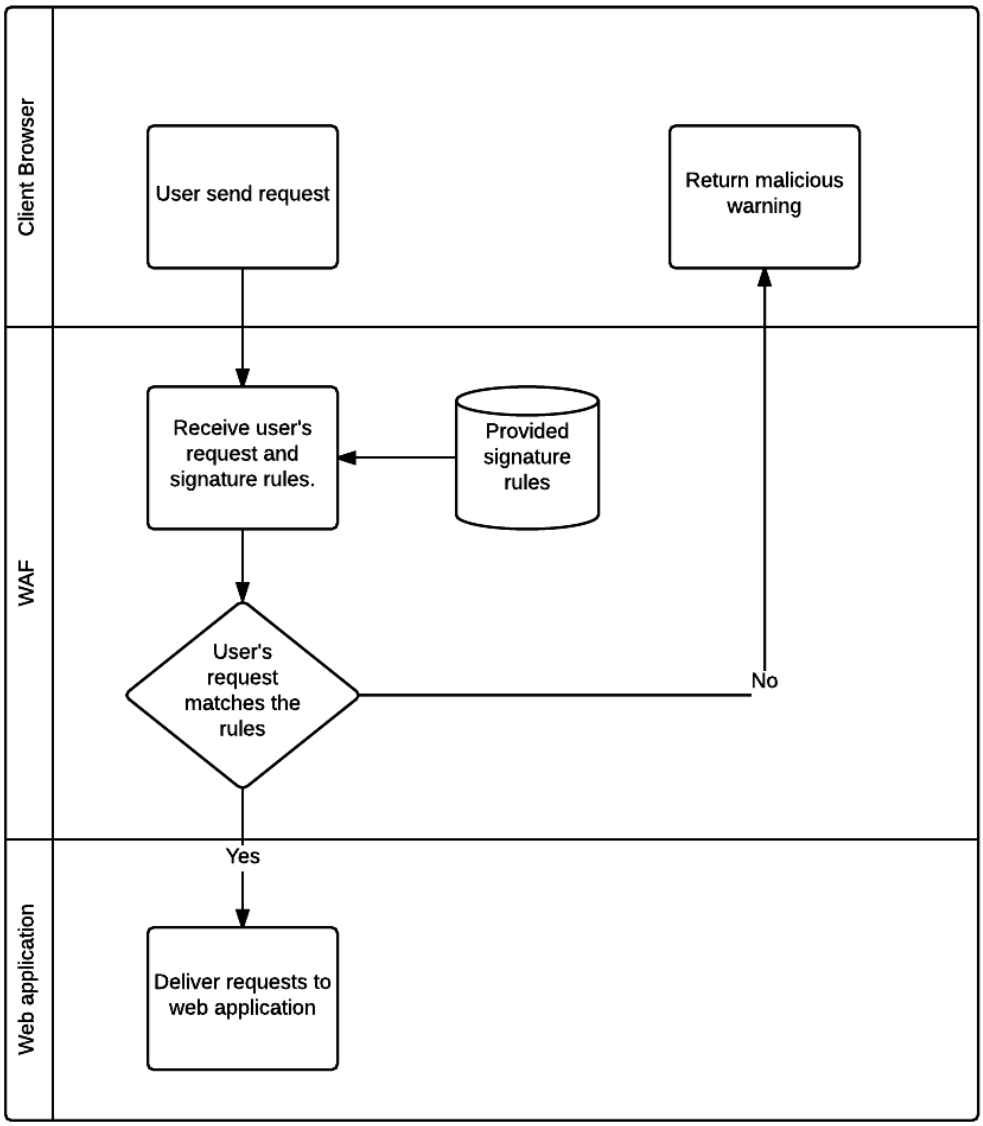

Figure 3. Flow Chart of Signature detection Module

2. Anomaly detection. The total detection procedure based on signature method is describe as flow chart shows (Figure4). The basic logic is: There are three different modes in sequential order: in training mode, User will send different HTTP requests continuously without any disturbing of WAF by clicking links, submit forms, etc. Note here all requests should be normal without malicious codes. The properties of each request will be stored in the database; next, in generating files mode, administrator will trigger this mode, stop training data, process data, set malicious rules and transfer it to detection mode; In the detection mode, new requests will be sent from users, including normal and malicious requests. The properties of each request will be recorded, processed and matched with training rules; if it matches, the request will be rejected and return warning, while not matches, the request will be sent to the web application to get the response. As the project description shows, the detection will happen as follows:
   1) Drop requests that the request exceeds that maximum number;
   2) Drop all values that the standard deviation are not part of mean+-3*sd

3) Drop requests with parameters containing characters from the sets.

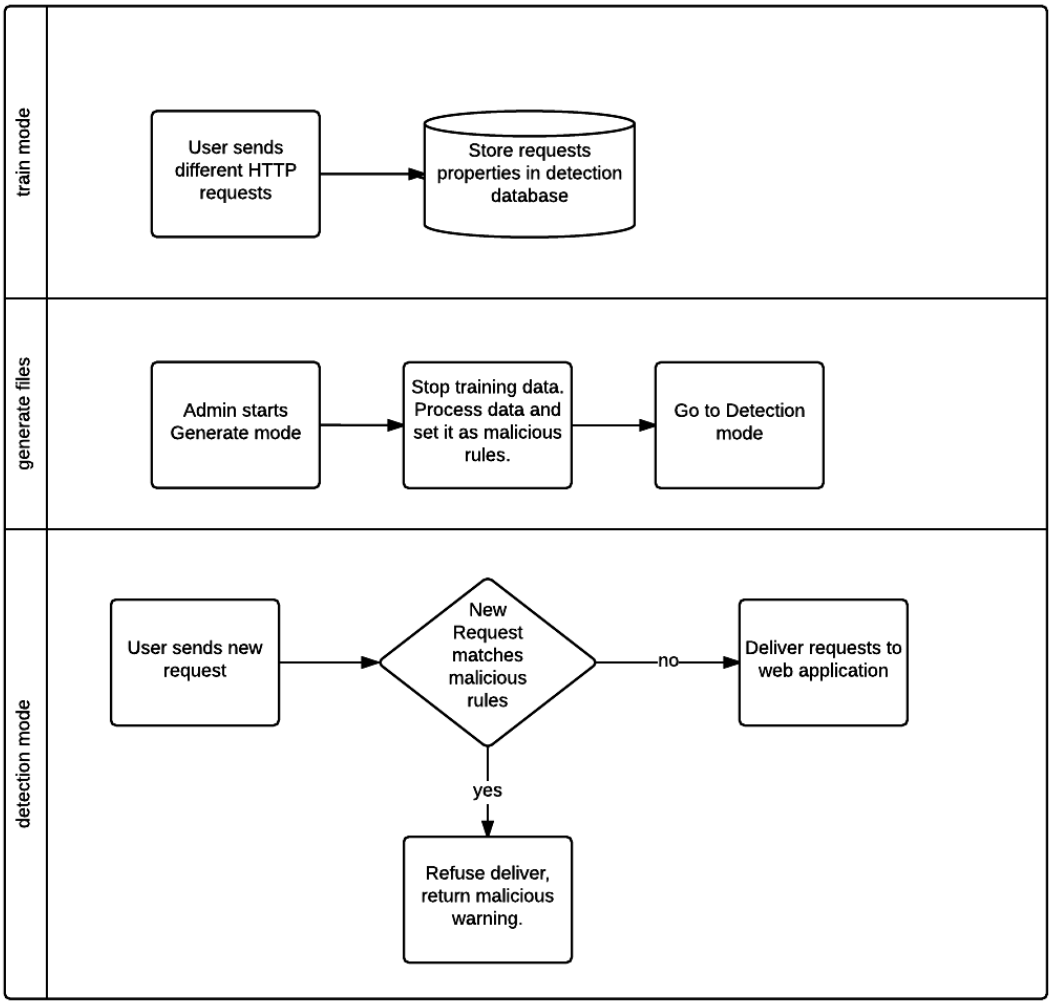

Figure 4. Flow Chart of Anomaly detection Module

Test Website Design:

The test website (Figure5), which behaviors as a blog website, is written in PHP5.4 and HTML5. The totally website consists of two main parts, user-side website for normal users and admin-side website for administrator management.

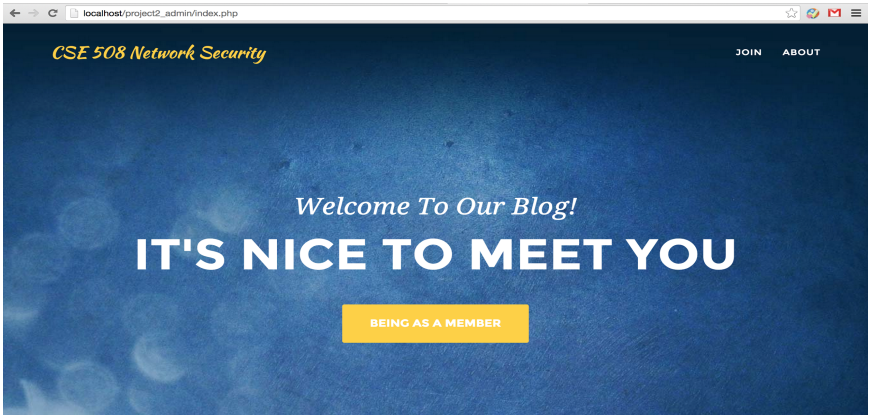

Figure 5. Test Website Home Page

In user side, one user could signed in, signed out, signed up his account (Figure 6); user could also view/create/edit/delete his blogs; and view/add short comments under each blog (Figure 7).

In administrator side, the administrator could manage all contents of the website (Figure8), including add/edit/delete website's users/blogs/comments.

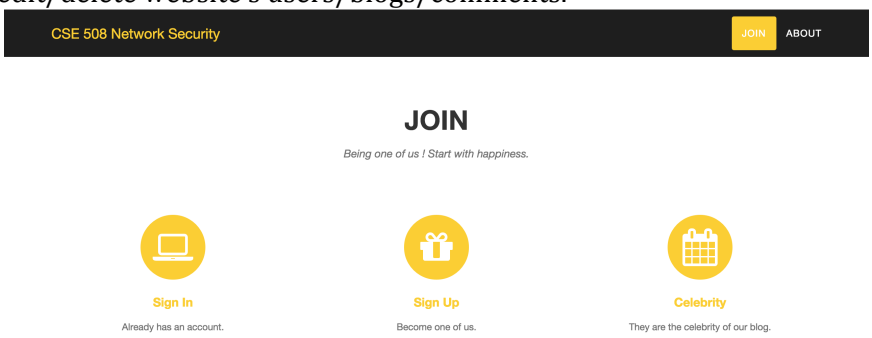

Figure 6. User Join Page

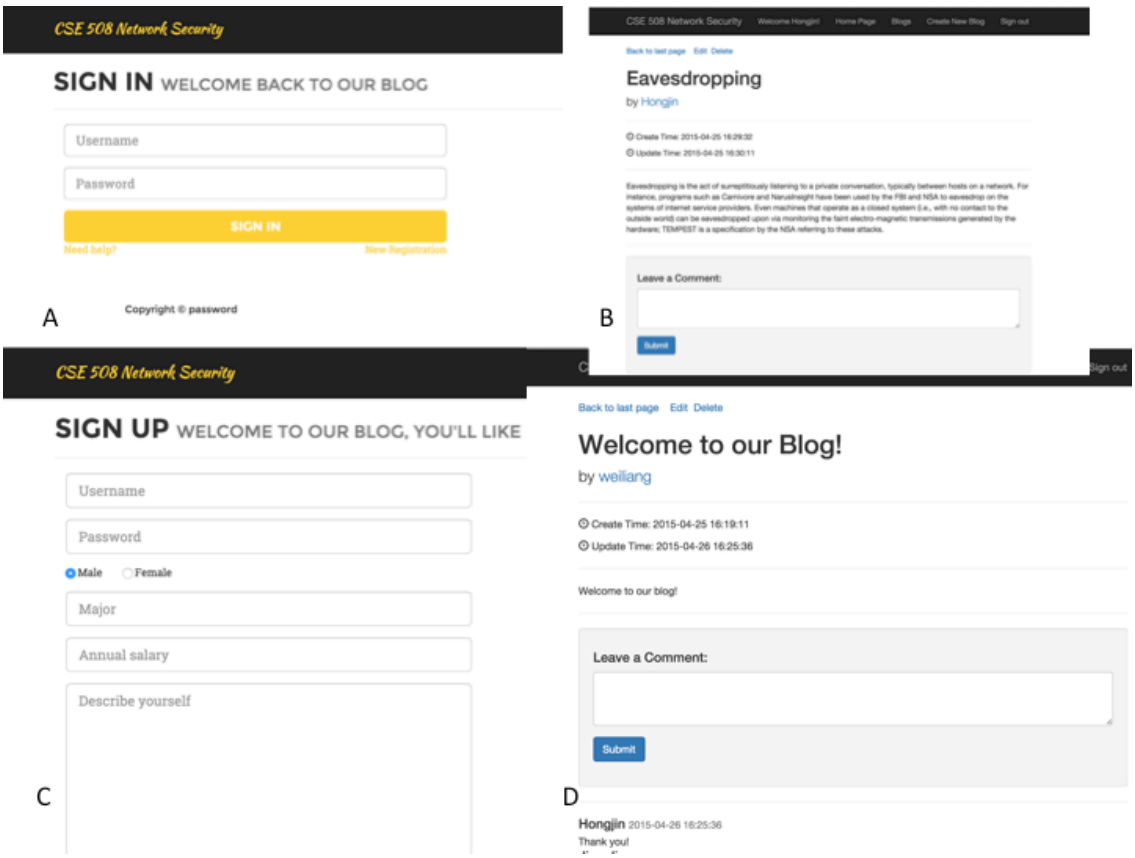

Figure 7. User-Side Webpages. A) Sign In Page; B) Individual Blog Page; C) Sign Up Page; D) Blogs Management Page.

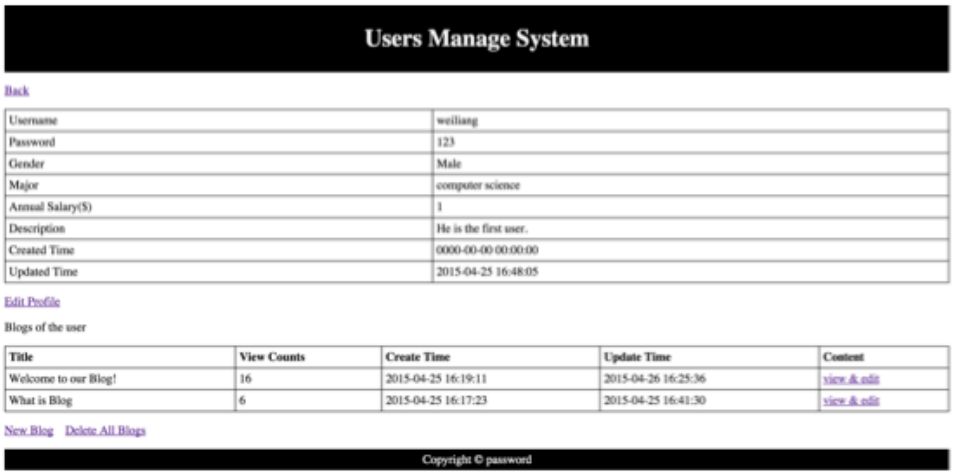

Figure 8. Admin-Side Webpages. Here shows the total user manage system for users, blogs and comments

For both user-side and admin-side websites, all data manipulations (Create, Read, Update and Delete) will be directly linked to the local database. This local database belongs to the test website, which is also the target of SQL injection attacks. (Note that here the database is used to manage data for tested web application website, not for apache WAF). For testing purposes, we make the test website completely invulnerable, which means many methods for security and validation purpose in *PHP* language like *mysqli_real_escape_string()* will not be included. The test website's front-end work is combined with Bootstrap.

Test Results:

1. Signatures test.

We use Firefox add-on Rest Easy [9] to generate Http Request to test the signatures detection. And our test cases are for these following attacks:

1. XSS attack - Get parameter contains “<script>”.
2. SQL injection - Get parameter contains “select union all”.
3. User-Agent contains ‘bot’’.
4. POST parameter “foo” contains “../../../../”.

See Figure 9 is a normal response from our test web application.

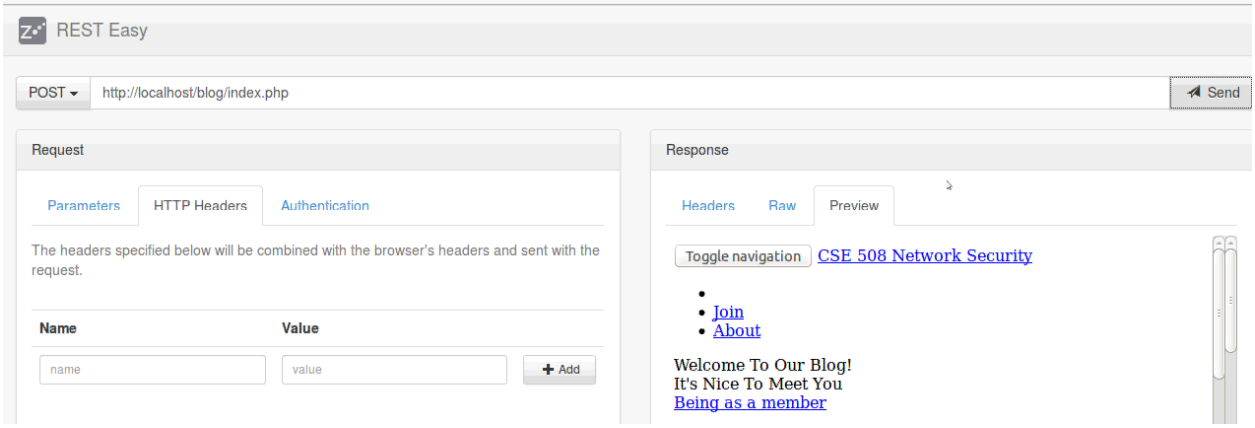

Figure 9: Normal test response.

(1) Test XSS attack. See Figure 10. We will check all GET parameters according to the rule:

REQUEST_METHOD:GET,PARAMETER:*,CONTAINS:"javascript" /*Detect XSS Attack*/.

If the parameter’s value contains “javascript”, we’ll consider this request contains XSS attack and then block it.

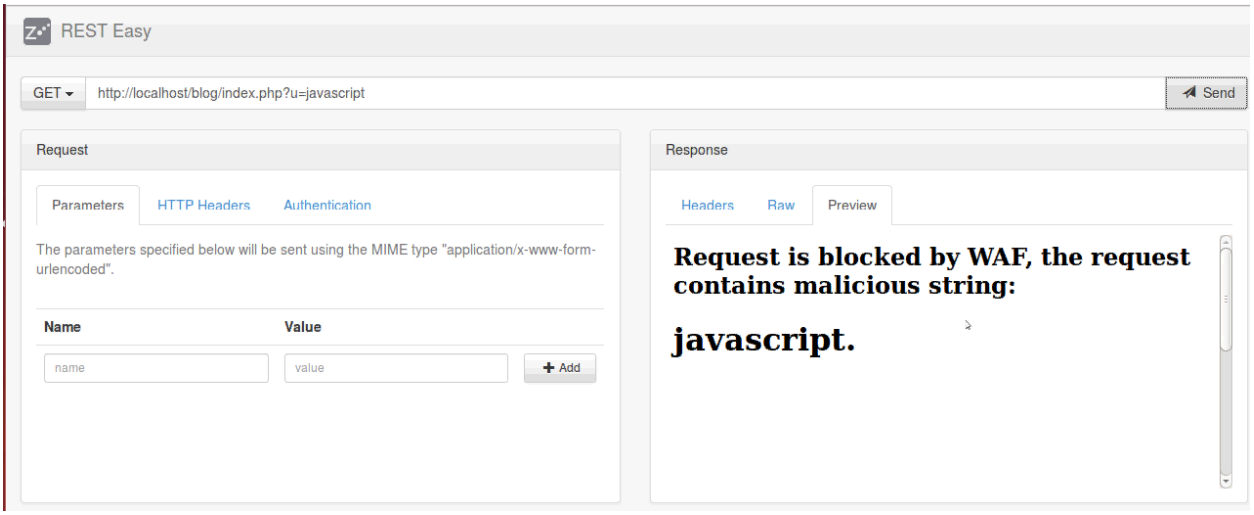

Figure 10: XSS attacks detection.

(2) Test SQL injection attack. See Figure 11. We will check all GET parameters according to the rule:

REQUEST_METHOD:GET,PARAMETER:*,CONTAINS:"select" /*Detect SQL Injection*/

If the parameter's value contains "select", we'll consider this request contains SQL injection attack and then block it.

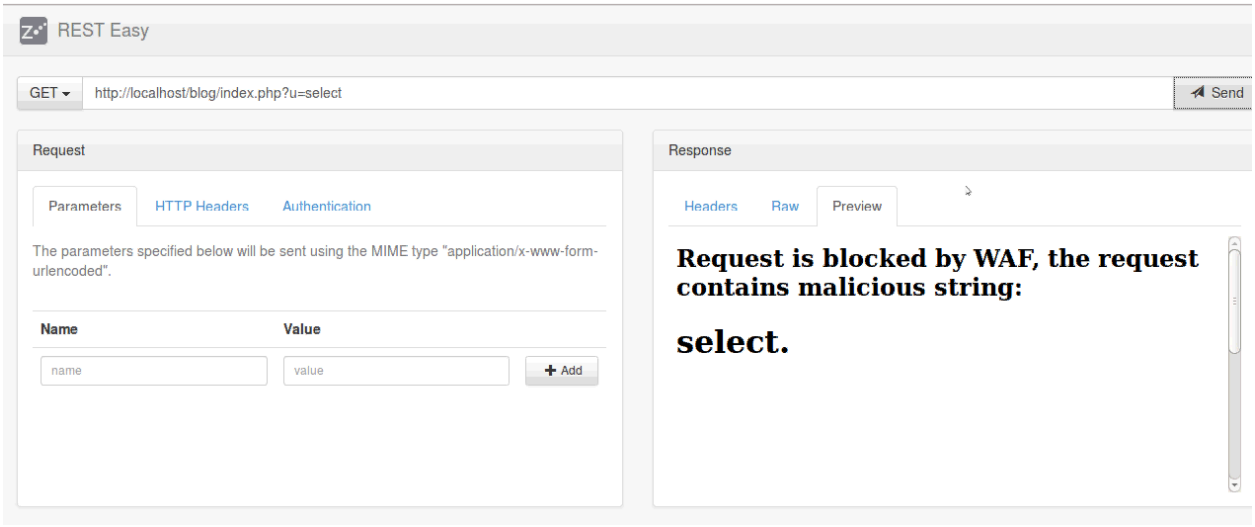

Figure 11: SQL injection attack detection.

(3) Test user-Agent contains 'bot''. See Figure 12. We will check all Header parameters according to the rule:

HEADER:User-Agent,CONTAINS:"bot"

/*Deny all requests from hosts that identify themselves as bots*/

If the parameter's value contains "bot", we'll consider this request is from user-agent, and then block it.

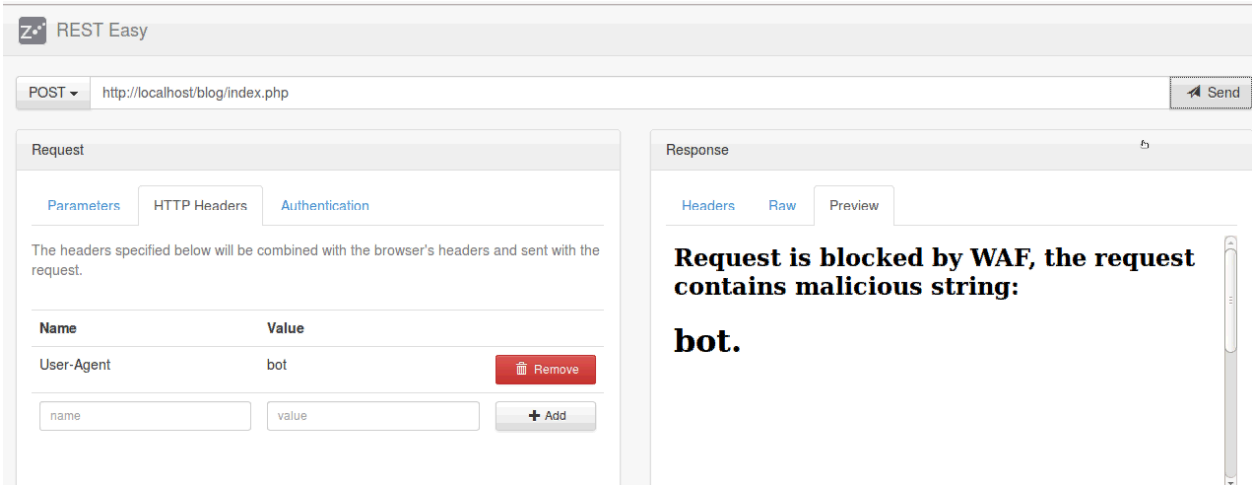

Figure 12: User-Agent attack detection.

(4) POST parameter "foo" contains "../../../../". See Figure 13. We will check all POST parameters according to the rule:

EQUEST_METHOD:POST,PARAMETER:foo,CONTAINS:"../../../../"

/*Search the parameter "foo" of POST Requests for possible Directory Traversal vulnerabilities*/

If the parameter's value contains "../../../../", we'll consider this request is the post dir travel attack, and then block it.

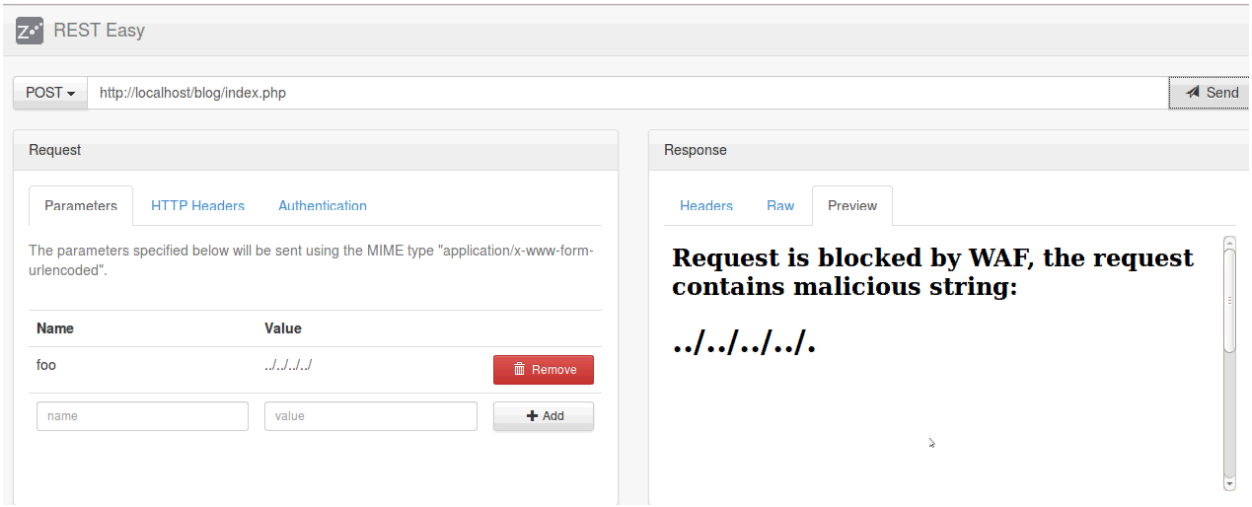

Figure 13: Post dir travel detection.

2. Anomaly Detection. In Train Mode, the WAF will not block any request, all request data (parameter number for each page; parameter value length; parameter value character set) will be stored in database. Then if the administrator wants to stop training and generate profile, he can choose to change mode to generate profile. After the WAF_Profile is successfully generated, WAF will change to detection mode. In detection mode, our WAF blocks the request that anomaly with the data in our training mode.

We use our blog website to test this method. Our test cases are for these following attacks:

1. Exceed maximum number of parameters of the whole website.
2. Exceed maximum number of parameters for current page.
3. Exceed maximum length of specific parameter for specific page.
4. Anomaly character set.

(1). Test exceeding maximum number of parameters of the whole website. See Figure 14. After we train, the WAF consider the maximum number of parameters of the normal website is 7. And when the user wants to visit an unknown webpage with exceeding number of parameters, our WAF will block the request and return the warning message.

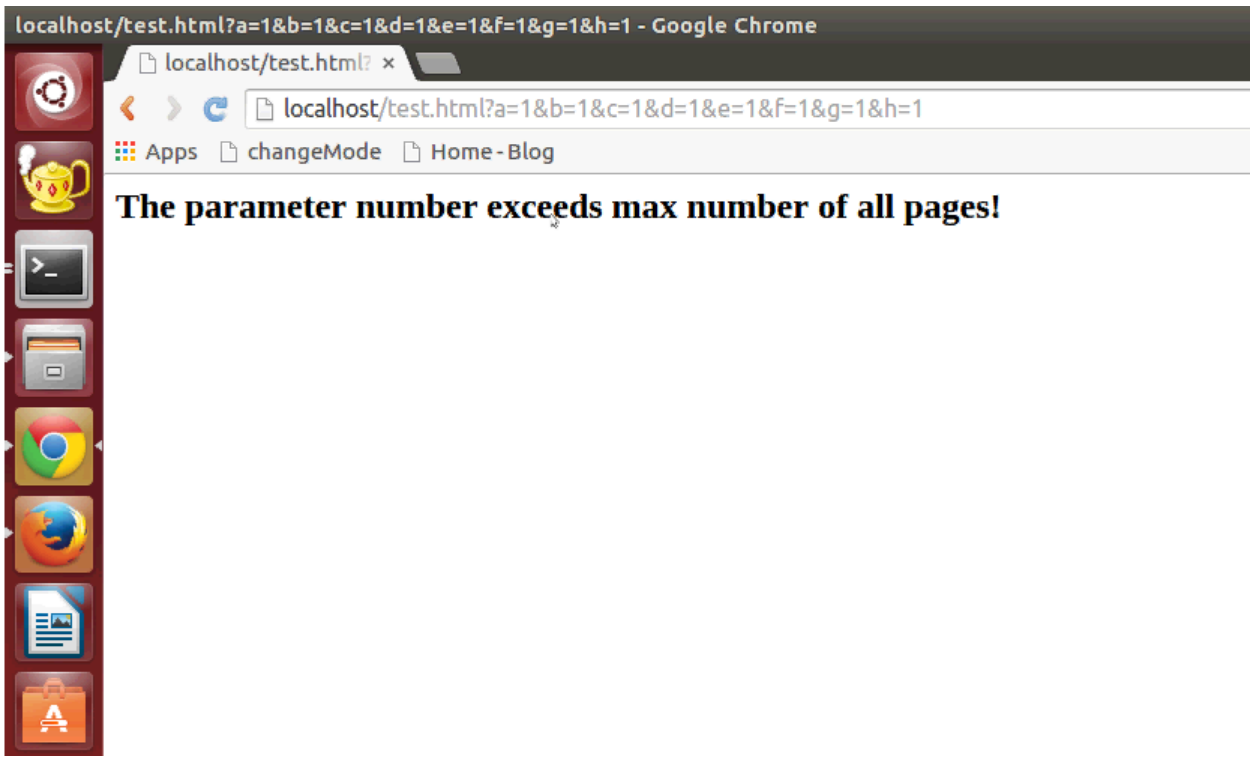

Figure 14: maximum number of parameters of the whole website detection.

(2). Test exceeding maximum number of parameters for current page. See Figure 15. After the training part, the WAF considers the max number of parameters of our test page user_main is 1. And when the user wants to visit a website with exceeding parameters, our WAF block this request and return a warning message.

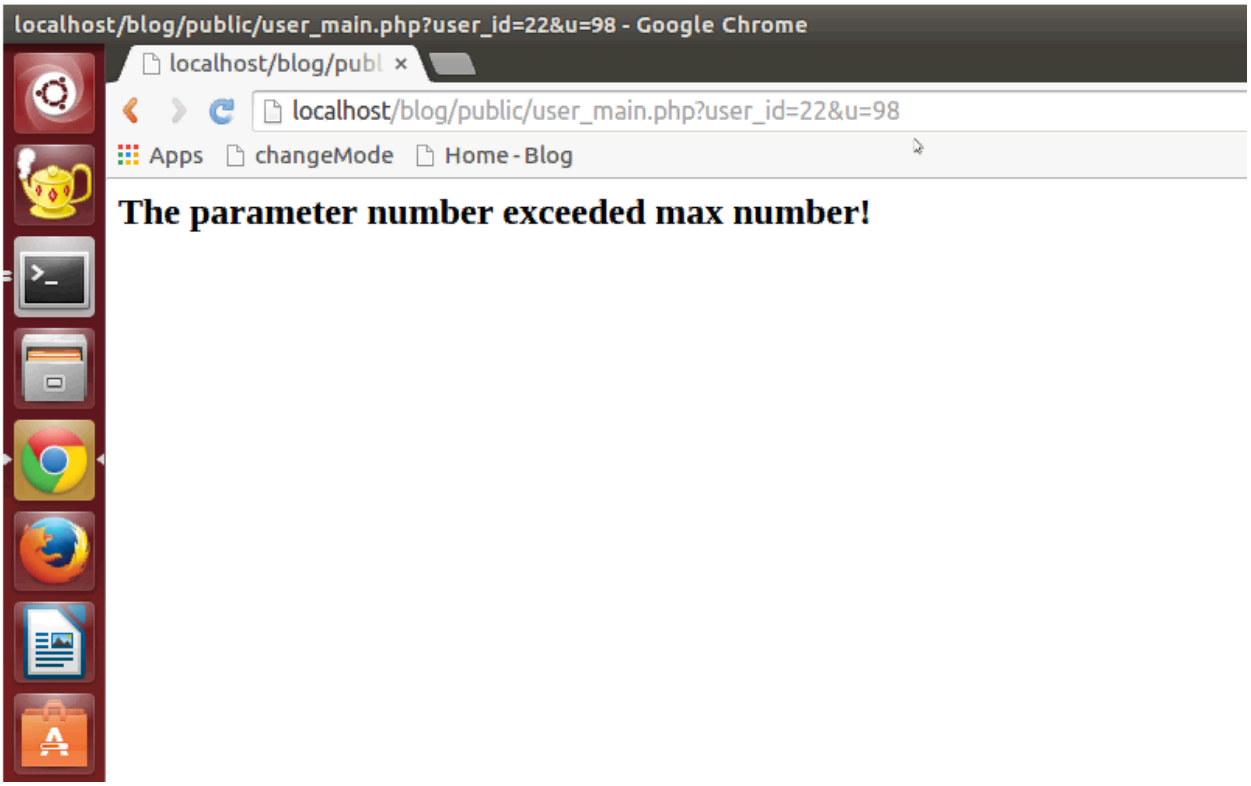

Figure 15: maximum number of parameters for current page detection.

(3). Test exceeding maximum length of specific parameter for specific page. See Figure 16. After training part, our WAF consider the maximum length of user_main page's parameter "user_id" with the normal length less than 3. When the user's requests with the user_id parameter longer than 3, our WAF will consider it as anomaly operation and block the request. Then return a warning message to the user.

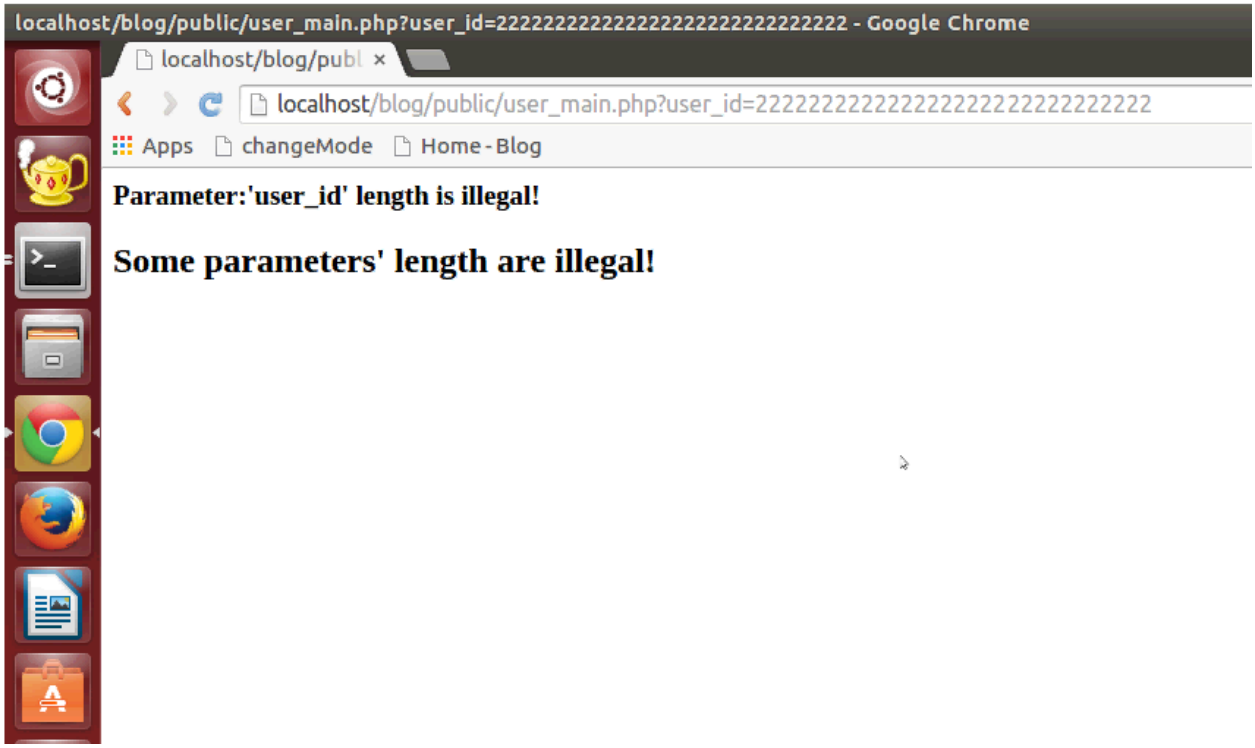

Figure 16: Maximum length of specific parameter for specific page detection.

(4). Test anomaly character set. See Figure 17. After training part, our WAF considers the charset in user_signup webpage are all digits and letters. And when the user enter an anomaly character, such as "@", our WAF will consider it as anomaly operation, and block the request. Then return a warning message to the user.

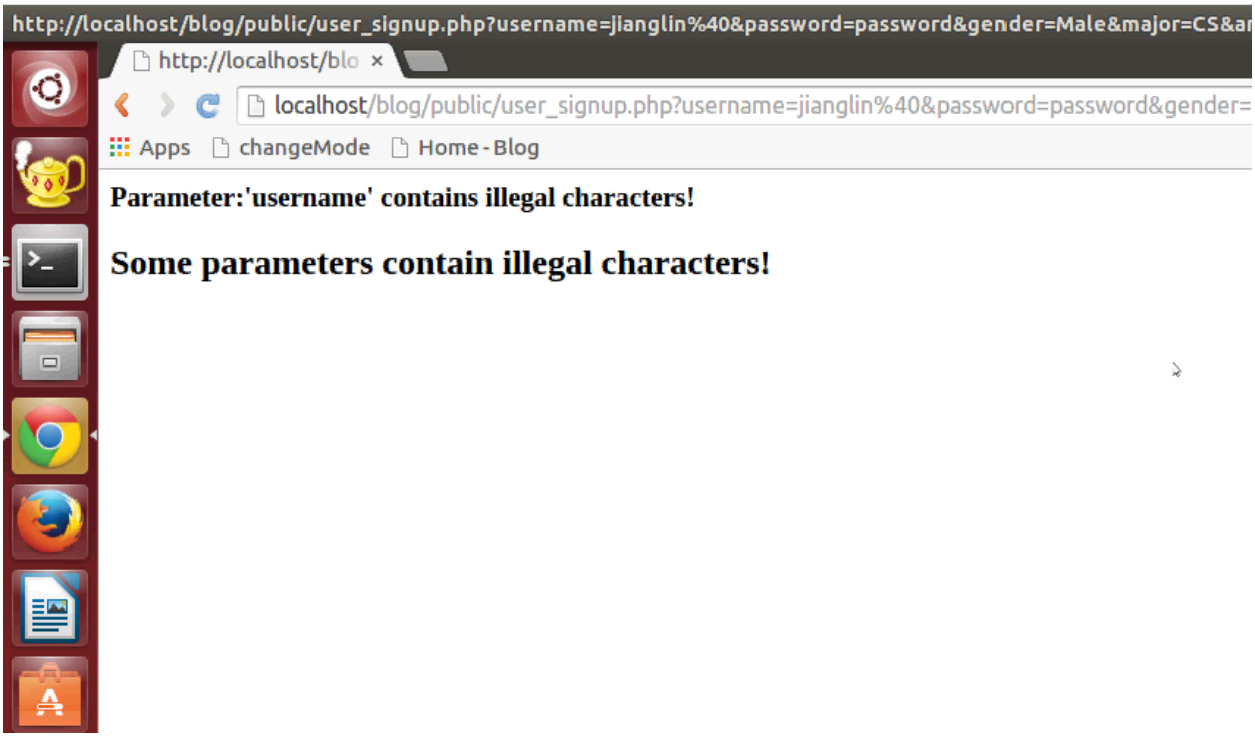

Figure 17: Anomaly charset detection.

Discussion & Conclusion

In this project, we implemented a web application firewall based on building modules on two detection methods: signature and anomaly detection. In signature detection part, we set almost all the conditions we know about to block the malicious request. We analysis the requests with GET, POST and HEADER parts with our malicious rules. And then block the requests that match the malicious rules in our signature detections. In anomaly detection, we save the legal operations into database. When trained data are large enough, the administrator changes the mode from training to generating file, which trains the data that we stored before to be our rules. After this, the generating file would turn into the detection mode. When a new request comes from user, the WAF will check the request with the rules. If the requests match the rules, the requests will be delivered to the web application. Otherwise the WAF will reject the request and return a warning to the user.

In order to implement and test all the function, vulnerable test website was developed, and databases for modules and test website are built. By testing different normal and malicious requests from test websites using basic web operations like clicking links, submit forms, etc., WAF will recognize malicious request to reject and allow access to web application if it is normal request. The experiments show that the good property of WAF recognition.

While our web application firewall works well by defined settings, limitations from methods themselves may further restrict the use of the WAF. The reasons are the lack of logics of these two simple methods. For example, request like *REQUEST_METHOD: GET,PARAMETER:*, CONTAINS:* "update" may exist both in normal and malicious request; if WAF put this request in the blacklist rules, it will have a high false positive error rate. Actually, majority of potential "markers" should be considered in this method due to its high false positive error rate, like "*UNION*", "*INSERT*", "*SELECT*", etc., which may also be used for malicious purposes. For fully detection rules, see "signature reference" at end of the report. For the second method, the accuracy of malicious request detections will highly depend on the requests in training session; suppose by mistake the requests are made malicious in training session, it will disturb or even destroy the detection procedure. Although these intrinsic limitations of the methods greatly restrict the application of WAF, they uncovered basic logic of the WAF used in practical conditions. Through successful implementation of the WAF required in this project, we gained more knowledge about relative fields involving WAF development and relative security issues with tightly team cooperation.

Acknowledgements:

We would like to thank Professor Nick Nikiforakis for his expert advice and encouragement throughout this difficult project, as well as TA of this course for necessary assistance.

References:

-Report references:

[1] https://www.owasp.org/index.php/Web_Application_Firewall.

[2] Luis F. Medina (2003). The Weakest Security Link Series (1st ed.). IUniverse. p. 54. ISBN 978-0-595-26494-0.
[3] http://en.wikipedia.org/wiki/Application_firewall.
[4]"Same Origin Policy - Web Security. W3.org.". Retrieved on November 4, 2014.
[5] http://en.wikipedia.org/wiki/Robots_exclusion_standard
[6] http://en.wikipedia.org/wiki/Uniform_resource_identifier
[7] https://www.acunetix.com/blog/articles/preventing-xss-attacks/
[8] http://httpd.apache.org/docs/2.4/developer/modguide.html
[9] https://addons.mozilla.org/en-Us/firefox/addon/rest-easy/

-Module Code and Installation References:

Apache2, PHP5, MySQL install and configuration
(we are using Apache2.4.7, PHP5.5.9 and MySQL5.5.41):
https://www.howtoforge.com/installing-apache2-with-php5-and-mysql-support-on-ubuntu-12.04-lts-lamp
Apache Module setup and api usage:
http://httpd.apache.org/docs/2.4/developer/modguide.html
Connect MySQL using C program API: http://www.cyberciti.biz/tips/linux-unix-connect-mysql-c-api-program.html

-Web application - Blog website:

UI bootstrap framework:
http://startbootstrap.com/template-overviews/agency/
http://startbootstrap.com/template-overviews/blog-post/
http://startbootstrap.com/template-overviews/blog-home/

-Appendix A. Part contents from Signature Configuration File

*HEADER:User-Agent,CONTAINS:"<script>" /*Detect basic XSS ATTACK through User-Agent Header*/*
*HEADER:User-Agent,CONTAINS:"bot" /*Deny all requests from hosts that identify themselves as bots*/*
REQUEST_METHOD:GET,PARAMETER:,CONTAINS:"'" /*Detect SQL Injection*/*
REQUEST_METHOD:GET,PARAMETER:,CONTAINS:"select" /*Detect SQL Injection*/*
REQUEST_METHOD:GET,PARAMETER:,CONTAINS:"union all" /*Detect SQL Injection*/*
REQUEST_METHOD:GET,PARAMETER:,CONTAINS:"<" /*Detect XSS Attack*/*
REQUEST_METHOD:GET,PARAMETER:,CONTAINS:">" /*Detect XSS Attack*/*
REQUEST_METHOD:GET,PARAMETER:,CONTAINS:"javascript" /*Detect XSS Attack*/*
*EQUEST_METHOD:POST,PARAMETER:foo,CONTAINS:"../../../../" /*Search the parameter "foo" of POST Requests for possible Directory Traversal vulnerabilities*/*