CSE-508
NETWORK SECURITY

SCRIPT-NONCE MODULE FOR APACHE

PROJECT REPORT

SUBMITTED BY

SAGARDEEP MAHAPATRA – 108771077
ALOK KATIYAR – 108943744

INTRODUCTION

The goal of the project is to build an apache filter module that will prevent Cross Site Scripting attacks by identifying all the injected malicious scripts on a web page and preventing them from getting executed on the client's browser when the web page is loaded from the server. This also requires some support from the browser that facilitates enabling/disabling of Content Security Policy features. Currently, some versions of browsers like Firefox and Google Chrome browser have extensions for supporting Content Security Policy. We verify our results on Google Chrome browser version 28 by enabling the "Experimental WebKit" features.

This report provides a brief overview of XSS, Content Security Policy, Apache Filter modules and then on the different performance analysis and results of our experiments.

FEEDBACK FROM PROJECT REVIEW

Following was the feedback provided on some areas of the project during the review. We worked on these issues and have listed out our findings on the same.

- Division of html data in bucket brigades and its effect on the apache module
- Micro level performance analysis with the filter module inserted at different places in the filter chain
- Micro level performance analysis using different substring finding functions

CROSS SITE SCRIPTING ATTACKS (XSS)

Cross Site Scripting, (XSS) is a typical computer security vulnerability special found in web applications, where a malicious client can inject scripts into the web page, which can then be executed on other clients' browsers when they are downloaded from the server, thus compromising the secrecy of their data.

Cross Site Scripting takes advantage of the "Same Origin Policy" which permits scripts running on pages originating from the same site (a combination of scheme, hostname, and port number) to access each other's methods and properties with no specific restrictions, but prevents access to most methods and properties across pages on different sites. This mechanism bears a particular significance for modern web applications that extensively depend on HTTP cookies to maintain authenticated user sessions.

XSS attacks can be broadly classified into two categories:

Non-Persistent XSS Attacks:

These are also called as Reflected XSS Attacks. The attack is delivered mostly by an email or a neutral website by providing a URL which has a link to the trusted website, but with the XSS vector attached to the end. The XSS vector is mostly a script that is required by the attacker to be executed on the client's browser and compromise some or most of the private information of the client.

Persistent XSS Attacks:

These are also called as stored XSS attacks and are considered even more dangerous than the non-persistent XSS attacks, because they are stored on the server permanently and are displayed to all the clients when they request information from the server.

CONTENT SECURITY POLICY

Content Security Policy is a declarative policy that lets the authors (or server administrators) of a web application informs the client about the sources from which the application expects to load resources.

A web application can declare that it only expects to load scripts from particular trusted sources and this can prevent the XSS attacks. There are various ways to hint the clients about the trusted sources so that they can successfully detect and block the malicious scripts injected into the application. In the project, we implement this functionality by using script nonce, which is a randomly generated number appended to the tags of all trusted scripts.

To take advantage of CSP, a web application opts into using CSP by supplying a Content-Security-Policy HTTP header. Such policies apply to the current resource representation only. To supply a policy for an entire site, the server needs to supply a policy with each resource representation.

SAMPLE WEB-PAGE WITH THE INJECTED SCRIPT

<script script-nonce="**some secret value**">
/* Valid script here */
</script>

<script>
/* evil injected script here */
</script>

SAMPLE PAGE AFTER APACHE MODULE REPLACES THE NONCE

<script script-nonce="**some fresh random value**">
/* Valid script here */
</script>

<script>
/* evil injected script here */
</script>

In the above scenario, when the web page gets loaded from the server, only the first script is identified as a valid script and is executed on the client's browser and the second script is not, because it does not have the valid script nonce.

SCRIPT NONCES

directive-name = "script-nonce"
directive-value = nonce
nonce = 1*(<VCHAR except ";" and ",">)

If the policy contains a script-nonce directive, then the server generates a random fresh value each time a client requests the page. Thus, the script nonce values are always different and it becomes impossible for a malicious attacker to guess its value.

When enforcing the script-nonce directive

a) Whenever the user would execute a script from a script element in the web page, it does not execute it unless the script nonce value is non-empty and matches (case sensitive match) the one provided in the HTTP header and the Meta directive for the CSP, if present.

b) Whenever the user agent would execute a script from an inline event handler, instead the user agent must not execute the script.

c) Whenever the user agent would execute script contained in a java script URI, instead the user agent must not execute the script.

APACHE FILTER MODULE

The apache filter modules are of different types. Since we are modifying the content of the html pages, hence, we build a content filter (filter type AP_FTYPE_RESOURCE). HTML content is received in data structures called bucket brigades. Bucket brigade is a doubly linked list of buckets that store the HTML data of the web pages.

Our filter module collects data from the bucket structures and searches for the placeholder for the script nonce and replaces that with a 32-bit random number. This process is repeated until all the bucket brigades for the web page are exhausted. The final list of bucket brigades as a whole contains the html data of the web page with the script nonce value replaced with the random value. Finally, we pass the bucket brigade to the next filter in the chain.

DEALING WITH SPLIT NONCE SCENARIO:

Consider the following scenario, where the html content is split between the buckets in the following manner:

[…. script-nonce = “abcdhg] → [hjshd” ….]

Bucket i Bucket i+1

Such scenarios will result in incorrect replacement of script nonce and hence will result in valid scripts not getting executed.

There are two solutions to the above problem:

1) Concatenate the html content from all the buckets into one single buffer and create one bucket and bucket brigade for the same. Now, since whole of the data is present in one buffer, the above issue of split nonce does not exist.

2) Bring consecutive buckets (say bucket i and bucket i+1) into a buffer and search and replace the random script nonce value. Then create a new bucket for only the first half of the buffer. Now, my new i is the previous i+1. This process is repeated till all the buckets are exhausted.

ISSUES WITH THE ABOVE APPROACHES:

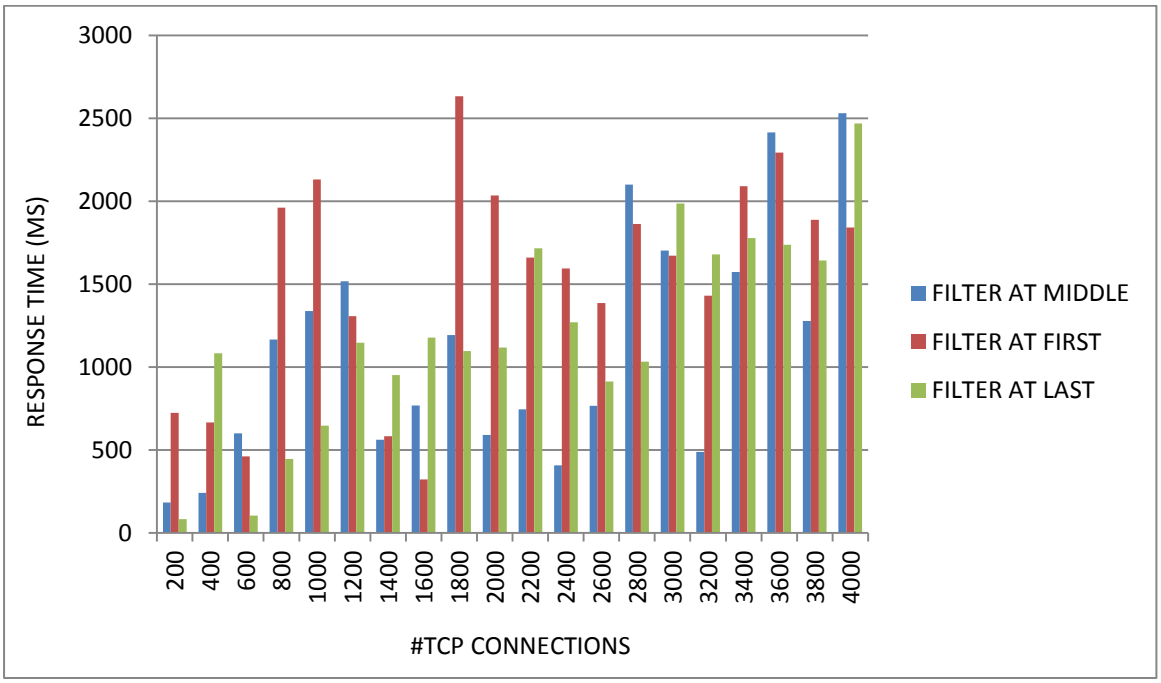
Both the solutions complicate the process of replacement of the script nonce and hence will reduce the performance of the apache server drastically.

However, we tested our simple filter module with various test scripts and never encountered the above mentioned scenario.

MICRO-LEVEL PERFORMANCE ANALYSIS

We inserted our filter module at different locations in the filter chain and analyzed its impact on the server's performance.

Observation: The performance was the best when the module was inserted at the middle.



Performance of filter at various locations

Another micro level performance analysis would be to use any other substring finding function in a string (instead of strstr that we are using in our module) and compare the results with strstr. However, since we have coded the module using the C API, we could not find any other substring finding functions in C.

MACRO-LEVEL PERFORMANCE ANALYSIS

HTTPERF

Standardized tool to measure the performance of http server

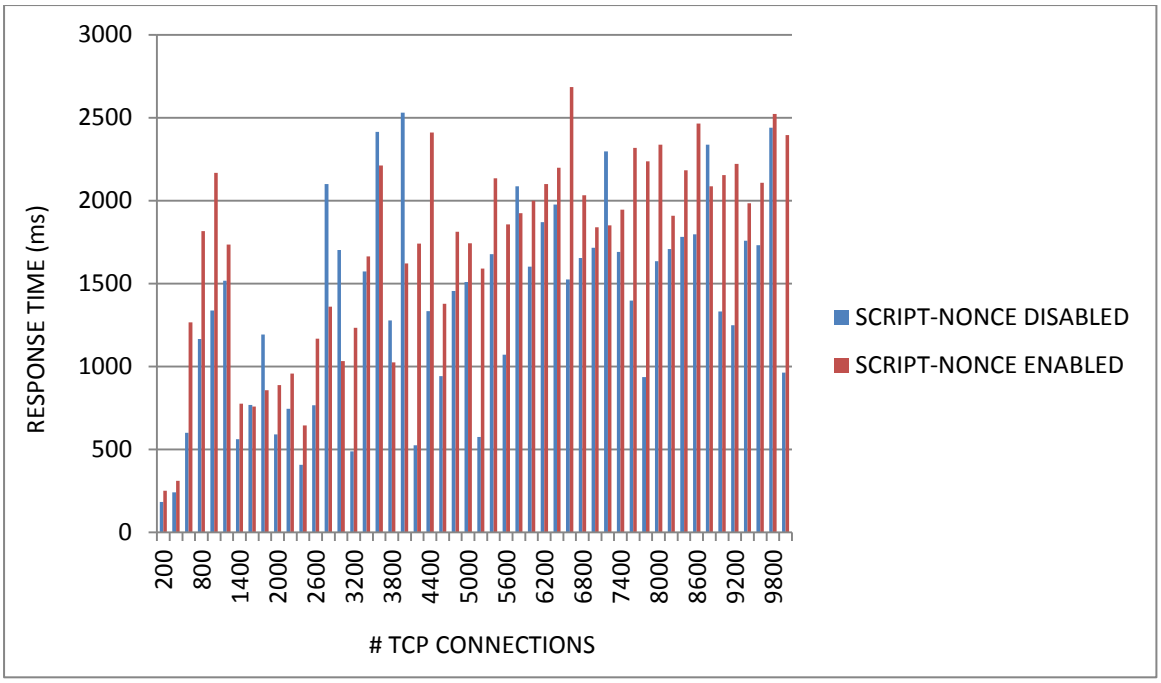
PARAMETERS

- server : IP of the machine on which the server is hosted
- uri : link to the php page requested
- num-con : number of TCP connections to the server
- num-call : number of requests/connection
- rate : number of requests/second sent from client to server

OUTPUT

- Response time per request
- Data transfer time from the server to the client per connection

RESULTS



We observe that the apache server slowed down by almost 25% on inclusion of the new filter

REFERENCES

[1] Base Code referred from here http://svn.apache.org/repos/asf/httpd/httpd/branches/2.2.x/modules/experimental/mod_case_filter.c

[2] http://www.apachecon.com/2007/notes/t02-e.pdf

[3] http://apache-http-server.18135.x6.nabble.com/Apache-2-2-Output-Filter-simple-C-example-td4828773.html

[4] http://www.apachetutor.org/dev/smart-filter

[5] http://linux.about.com/od/ubusrv_doc/a/ubusg25t08.htm

[6] http://httpd.apache.org/docs/current/mod/mod_filter.html

[7] http://httpd.apache.org/docs/2.2/programs/apxs.html

[8] http://www.penguinpowered.org/documentation/apache2_modules.html

[9] http://httpd.apache.org/docs/2.2/mod/mod_so.html

[10] http://chandpriyankara.blogspot.com/2012/02/start-with-apache-httpd-module.html

[11] https://developer.mozilla.org/en-US/docs/Security/CSP

[12] http://httpd.apache.org/docs/2.4/mod/mod_example.html#using

[13] https://en.wikipedia.org/wiki/Cross-site_scripting

[14] http://www.apachetutor.org/dev/