CSE508: Network Security
Project Report
Script Nonce Module for Apache

Ameya Hemant Patnekar, Ganesa Thandavam Ponnuraj, Lalit Ramesh Sirsikar
Date: May-23,2013 **Professor:** Prof. Rob Johnson

Introduction:

Cross-site scripting (XSS) is a type of computer security vulnerability typically found in Web applications. XSS enables attackers to inject client-side script into Web pages viewed by other users. A cross-site scripting vulnerability may be used by attackers to bypass access controls such as the same origin policy.

Generally, two types of attacks are possible, reflected and stored XSS attacks. One of the example for reflected attacks occur when a user clicks on a malicious URL, which gets directed to the server; the server accepts the URL supposing the user as genuine and responds back. The response carries with itself the malicious scripts which then get executed on the user's browser. The browser also accepts this based on the same origin policy security concept. An example of a stored XSS attack is as follows: Suppose there is a comment section on some website. Now if a user injects a script in this text-field, it then goes to the server and gets stored in the Data base. When some other user logs on to that page again, the script gets loaded as a part of page. And as the malicious script is coming from the server, the victims browser executes it due to same origin policy and thus the attack is accomplished.

This report first states the problem, and then proposes solution to the problem. Our implementation is then described in detail followed by benchmarking results.

Problem Description:

The problem with the above attacks is that malicious scripts can get executed on client's browser. There are specific methods to control this from client's side, but a more general scene would be to tackle these attacks from the server side itself. The server attempts to prevent this attack from its side by inserting nonces into script values (Content Security Policy script nonce). The client then executes only those scripts that have valid nonces in them. Here, our implementation is described to demonstrate this on Apache server.

Implementation:

The implementation to prevent XSS attacks on the server side has been made as follows. Apache comes with a module called *mod_substitute* which when configured and stated in httpd.conf allows for replacement of a string with another. The syntax of doing that is as follows:

```
<Location />AddOutputFilterByType SUBSTITUTE text/html
Substitute s/foo/bar/ni
</Location >
```

which substitutes each occurrence of string "foo" with string "bar" and it also accepts any optional parameters that the user wants to give - 'n' stands for treating the pattern as a fixed string as opposed to a regular expression, and 'i' performs a case-insensitive match. Exploiting this feature of *mod_substitute*, we have defined our own module and loaded it by specifying in httpd.conf file. The syntax of our module is as follows:

```
<Location />AddOutputFilterByType ADD-SCRIPT-NONCE text/html
Addscriptnonce s/server-secret/
</Location >
```

Algorithm: We crawl through the entire website with a python module which takes as input a set of file-names and replaces each occurrence of string "<script" with "<script nonce=server-secret", thus each page

now has a script-nonce attached with it, which at present is servers secret value. This value for nonce during pre-processing is configurable, and can be modified periodically by the web-admin.
Next, our module is enabled through httpd.conf by specifying the following:

```
LoadModule script_nonce_module modules/mod_script_nonce.so
```

The module acts by attaching a script-nonce header field in the response URL and filling it with random nonce value (we use *apr_generate_random_bytes* for this and convert all hexadecimal values to alphanumeric characters). The module at the same time also fills all the occurrences of the server-secret given in httpd.conf with the value generated. The page when delivered to client (that understands CSP1.1 nonce values) checks if the nonce value matches; the script is executed if the nonce value matches, else it is not executed.

Defence:
Stored XSS:

Every valid script delivered by the server contains a script nonce with a random value. And the same nonce value is also specified in the CSP Header. So, even if any attacker succeeds to inject his malicious script into the page, it won't be executed by the browser as it will not contain valid script nonce.
Reflected XSS:

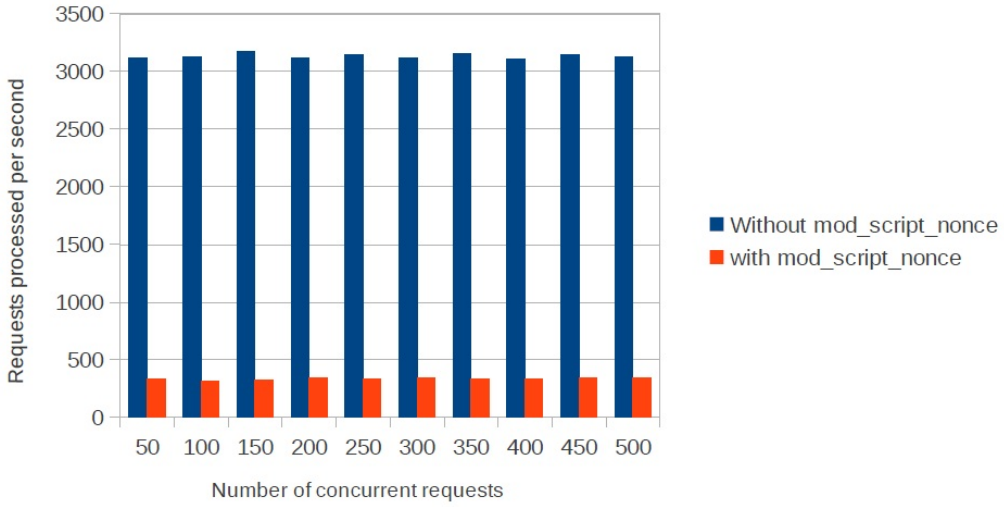
Chrome uses an Anti-XSS filter, based on static analysis, which attempts to detect XSS. If it detects such an attempt, it filters out the injected code, and effectively stops the on-going attack. We tested the following specially crafted URL which contain Script in it. Although it worked with Internet explorer and Mozila firefox, it didn't work with Chrome due to Chrome's Anti-XSS filter. Here are some examples:

http://securitee.tk/files/chrome_xss.php? a=<script>alert(1);&b=bar
http://securitee.tk/files/chrome_xss.php? a=<script>alert(1);/*&b=*/</script>
http://securitee.tk/files/chrome_xss.php? a=<script>/*//&b=*/alert(1);</script>
http://securitee.tk/files/chrome_xss.php? a=<script>void('&b=');alert(1);</script>

Performance:

The performance was measured with Apache's benchmarking tool called 'ab'. The tool comes as a module with Apache which takes as input 3 parameters - total number of requests, number of concurrent requests and URL. It then gives a complete output stating various time values (Connect, Processing, Waiting, Total). We wrote a python script for testing mean value results for a page containing 80 script-nonce tags. The module runs the command "ab -n 1000 -c 50+50*i http://www.sirsikar.com/benchmarkResults" for i in range(1,11) and takes the mean over all the values obtained. The number of iterations can be user-specified. The data here is for 5 iterations:

Case	Total Number Of Request	Number of Simultaneous Requests	Number of requests per second Without Nonce	Number of requests per second With Nonce
1	1000	50	3115.37	331.248
2	1000	100	3132.208	315.944
3	1000	150	3172.228	327.4
4	1000	200	3117.508	339.14
5	1000	250	3150.548	337.932
6	1000	300	3114.326	339.43
7	1000	350	3153.028	337.454
8	1000	400	3109.476	336.076
9	1000	450	3145.006	339.49
10	1000	500	3126.9	340.498



References:

http://www.owasp.org/
http://www.w3.org/
http://www.apache.org/
http://httpd.apache.org/docs/2.2/mod/mod_substitute.html
http://httpd.apache.org/docs/2.2/programs/ab.html
http://www.cyberciti.biz/tips/howto-performance-benchmarks-a-web-server.html
http://www.ibm.com/developerworks/library/wa-secxss/
http://blog.securitee.org/?p=37