Different Rewrite Methods available with mod_rewrite for BLACKLISTING

To facilitating site security, the techniques presented in this E-Book will improve your understanding of the different rewrite methods available with **mod_rewrite**.

CHETAN SONI

(Cyber Security Specialist)

Email – chetansoni@live.com

Facebook – http://facebook.com/er.chetansoni

Twitter – http://twitter.com/iamchetansoni

About Me –

I am a social-techno-learner who believes in its own efficiency first and then implements with the suggestions of my strong and enthusiastic Team which helps me takes everything into its perfection level.

The young and dynamic personality has not only assisted in solving complex cases but has also played an instrumental role in creating awareness about Information Security and Cyber crimes.

I conducted more than 100 workshops on topics like "**Botnets, Metasploit Framework, Networking, Vulnerability Assessment, Penetration Testing, Cyber Crime Investigation, Cyber Forensics and Ethical Hacking**" at various institutions/Colleges/Companies all across the world.

Achievements –

1. Experience as System Administrator, Support Engineer, Network Engineer, IT faculty, Technical consultant.
2. Extensive Experience in **Red Hat Enterprise Linux**.
3. Experience in designing cable and wireless networks, network cabling such as STP, UTP, coaxial etc., installation and configuration of LAN, WAN and wireless networks with active components such as hub, bridge, routers, switches, modems, repeaters etc. break / fix engineering.
4. Energetic and self-motivated team player. Proven ability to work in tight schedule and both independent and team environments.
5. Extensive Experience in **Backtrack Operating System** which is a Linux based OS.
6. Analysis and Monitoring of Packets in a Wireless Network.
7. Published more than **50 E-Books** and **24 Tools** in **Seculabs – Online Digital Library** related to Hacking, Cracking, Backtrack, Metasploit, Digital Forensics, Wi-Fi Hacking, and Website Hacking.
8. Brand Ambassador of the year 2011 at **Secugenius Security Solutions**.
9. Published My Paper on "**Complete WordPress Security**" at **Packet Storm Security** Website which is a **Global Security Resource.**
10. Research Paper Published on **"Capturing of HTTP Protocol Packets in a Wireless Network"** in **IJECCE** (International Journal of Electronics Communication and Computer Engineering)
11. Got "**Best Speaker of the year – 2013″** Award in **Chakravyuh IT Conference** held at **IIT-Delhi.**

Professional EXPERIENCE

Working as a **Sr. Security Specialist** at **SECUGENIUS SECURITY SOLUTIONS,** LUDHIANA from June '2011 & **Sr. Author** at **Seculabs – Online Digital Library** from January 2012.

Different Rewrite Methods available with mod_rewrite for BLACKLISTING

Chetan Soni

It includes;

1. Blacklist via Request Method
2. Blacklist via the Request
3. Blacklist via the Referrer
4. Blacklist via Cookies
5. Blacklist via Request URI
6. Blacklist via the User Agent
7. Blacklist via the Query String
8. Blacklist via IP Address

1. Blacklist via Request Method

It evaluates the client's request method. When a client attempts to connect to your server, it directly sends a message indicating the type of connection it wishes to make.

There are many different types of request methods recognized by Apache like **GET, PUT, POST, DELETE, CONNECT, OPTIONS, TRACE, PATCH, PROPFIND, PROPPATCH, MKCOL, COPY, MOVE, LOCK, UNLOCK, VERSION_CONTROL, CHECKOUT, UNCHECKOUT, CHECKIN, UPDATE, LABEL, REPORT, MKWORKSPACE, MKACTIVITY, BASELINE_CONTROL, MERGE and INVALID**.

The most common methods are **GET** and **POST** requests, which are required for "**getting**" and "**posting**" data to and from the server.

GET Method –

```
/abc/form.asp?name1=value1&name2=value2
```

POST Method –

```
POST /abc/form.asp HTTP/1.1
Host: chetansonisecurityspecialist.com
name1=value1&name2=value2
```

To restrict the types of request methods available to clients, we use this block of Apache directives:

```
<IfModule mod_rewrite.c>
 RewriteEngine On
 ServerSignature Off
 Options +FollowSymLinks
 RewriteCond %{REQUEST_METHOD} ^(delete|head|trace|track) [NC]
 RewriteRule ^(.*)$ - [F,L]
</IfModule>
```

GET Requests	POST Requests
a. GET requests can be cached	a. POST requests are never cached
b. GET requests remain in the browser history	b. POST requests do not remain in the browser history
c. GET requests can be bookmarked	c. POST requests cannot be bookmarked
d. GET requests should never be used when dealing with sensitive data	d. POST requests have no restrictions on data length
e. GET requests have length restrictions	
f. GET requests should be used only to retrieve data	

2. Blacklist via the Request

It is totally based on client's request. When a client attempts to connect to the server, it sends a full **HTTP** request string that specifies the request method, request **URI**, and transfer-protocol version and the additional headers sent by the browser are not included in the request string.

Example:-

```
GET blog/index.html HTTP/1.1
```

Here is an example of sanitizing client requests by way of Apache's THE_REQUEST variable:-

```
<IfModule mod_rewrite.c>
 RewriteEngine On
 RewriteCond %{THE_REQUEST} ^.*(\\r|\\n|%0A|%0D).* [NC]
 RewriteRule ^(.*)$ - [F,L]
</IfModule>
```

3. Blacklist via the Referrer

Blacklisting via the **HTTP referrer** is an excellent way to block referrer spam, defend against penetration tests, and protect against other malicious activity.

The HTTP referrer is identified as the source of an incoming link to a web page. For example, if a visitor arrives at your site through a link they found in the Google search results, the referrer would be the Google page from whence the visitor came.

Unfortunately, one of the biggest spam problems on the Web involves the abuse of **HTTP** referrer data. In order to improve search-engine rank, spam bots will repeatedly visit your site using their spam domain as the referrer. The referrer is generally faked, and the bots frequently visit via HEAD requests for the sake of efficiency. If the target site publicizes their access logs, the spam sites will receive a search-engine boost from links in the referrer statistics.

Fortunately, by taking advantage of **mod_rewrite's HTTP_REFERER** variable, we can forge a powerful, customized referrer blacklist.

Here's our example:

```
<IfModule mod_rewrite.c>
 RewriteEngine On
 RewriteCond %{HTTP_REFERER} ^(.*)(<|>|'|%0A|%0D|%27|%3C|%3E|%00).* [NC,OR]
 RewriteCond %{HTTP_REFERER} ^http://(www\.)?.*(-|.)?adult(-|.).*$  [NC,OR]
 RewriteCond %{HTTP_REFERER} ^http://(www\.)?.*(-|.)?poker(-|.).*$  [NC,OR]
 RewriteCond %{HTTP_REFERER} ^http://(www\.)?.*(-|.)?drugs(-|.).*$  [NC]
 RewriteRule ^(.*)$ - [F,L]
</IfModule>
```

4. Blacklist via Cookies

Protecting your site against malicious cookie exploits is greatly facilitated by using Apache's **HTTP_COOKIE** variable.

HTTP cookies are chunks of data sent by the server to the web client upon initialization. The browser then sends the cookie information back to the server for each subsequent visit.

This enables the server to authenticate users, track sessions, and store preferences. A common example of the type of functionality enabled by cookies is the shopping cart. Information about the items placed in a user's shopping cart may be stored in a cookie, thereby enabling server scripts to respond accordingly.

Generally, a cookie consists of a unique string of alphanumeric text and persists for the duration of a user's session. Apache's mod_cookie module generates cookie values randomly and upon request. Once a cookie has been set, it may be used as a database key for further processing, behavior logging, session tracking, and much more.

Unfortunately, this useful technology may be abused by attackers to penetrate and infiltrate your server's defenses. Cookie-based protocols are vulnerable to a variety of exploits, including cookie poisoning, cross-site scripting, and cross-site cooking. By adding malicious characters, scripts, and other content to cookies, attackers may find and exploit sensitive vulnerabilities.

Here is an example that does the job:

```
<IfModule mod_rewrite.c>
 RewriteEngine On
 RewriteCond %{HTTP_COOKIE} ^.*(<|>|'|%0A|%0D|%27|%3C|%3E|%00).* [NC]
 RewriteRule ^(.*)$ - [F,L]
</IfModule>
```

5. Blacklist via Request URI

Use of Apache's **REQUEST_URI** variable is frequently seen in conjunction with URL canonicalization.

The REQUEST_URI variable targets the requested resource specified in the full HTTP request string. Thus, we may use Apache's **THE_REQUEST** variable to target the *entire* request string, while using the **REQUEST_URI** variable to target the actual request URI.

For example, the REQUEST_URI variable refers to the "**blog/index.html**" portion of the following, full HTTP request line:

```
GET blog/index.html HTTP/1.1
```

For canonicalization purposes, this is *exactly* the type of information that must be focused on and manipulated in order to achieve precise, uniform URLs.

Likewise, for blacklisting malicious request activity such as the kind of nonsense usually exposed in your server's access and error logs, targeting, evaluating, and denying malicious URL requests is easily accomplished by taking advantage of Apache's REQUEST_URI variable.

As you can imagine, blacklisting via **REQUEST_URI** is an excellent way to eliminate scores of malicious behavior.

Here is an example that includes some of the same characters and strings that are blocked in the upcoming 4G Blacklist:

```
<IfModule mod_rewrite.c>
 RewriteEngine On
 RewriteCond %{REQUEST_URI} ^.*(,|;|:|<|>|">|"<|/|\\\.\.\\).* [NC,OR]
 RewriteCond %{REQUEST_URI} ^.*(\=|\@|\[|\]|\^|\`|\{|\}|\~).* [NC,OR]
 RewriteCond %{REQUEST_URI} ^.*(\'|%0A|%0D|%27|%3C|%3E|%00).* [NC]
 RewriteRule ^(.*)$ - [F,L]
</IfModule>
```

6. Blacklist via the User Agent

Blacklisting via **user-agent** is a commonly seen strategy that yields questionable results. The concept of blacklisting user-agents revolves around the idea that every browser, bot, and spider that visits your server identifies itself with a specific user-agent character string.

Thus, user-agents associated with malicious, unfriendly, or otherwise unwanted behavior may be identified and blacklisted in order to prevent against future access. This is a well-known blacklisting strategy that has resulted in some extensive and effective user-agent blacklists.

Of course, the downside to this method involves the fact that user-agent information is easily forged, making it difficult to know for certain the true identity of blacklisted clients. By simply changing their user-agent to an unknown identity, malicious bots may bypass every blacklist on the Internet. Many evil "**scumbots**" indeed do this very thing, which explains the incredibly vast number of blacklisted user-agents.

Even so, there are certain limits to the extent to which certain user-agent strings may be changed.

For example, **GNU's Wget** and the **cURL command-line tool** are difficult to forge, and many other clients have hard-coded user-agent strings that are difficult to change.

On Apache servers, user-agents are easily identified and blacklisted via the **HTTP_USER_AGENT** variable.

Here is an example:

```
<IfModule mod_rewrite.c>
 RewriteEngine On
 RewriteCond %{HTTP_USER_AGENT} ^$                                          [OR]
 RewriteCond %{HTTP_USER_AGENT} ^.*(<|>|'|%0A|%0D|%27|%3C|%3E|%00).*                    [NC,OR]
 RewriteCond %{HTTP_USER_AGENT} ^.*(HTTrack|clshttp|archiver|loader|email|nikto|miner|python).* [NC,OR]
 RewriteCond %{HTTP_USER_AGENT} ^.*(winhttp|libwww\-perl|curl|wget|harvest|scan|grab|extract).* [NC]
 RewriteRule ^(.*)$ - [F,L]
</IfModule>
```

7. Blacklist via the Query String

Protecting your server against malicious query-string activity is extremely important. Whereas static URLs summon pages, their appended query strings transmit data and pass variables throughout the domain.

Query-string information interacts with scripts and databases, influencing behavior and determining results. This relatively open channel of communication is easily accessible and prone to external manipulation.

By altering data and inserting malicious code, attackers may penetrate and exploit your sever directly through the query string.

Fortunately, we can protect our server against malicious query-string exploits with the help of Apache's invaluable **QUERY_STRING** variable. By taking advantage of this variable, we can ensure the legitimacy and quality of query-string input by screening out and denying access to a known collection of potentially harmful character strings. Here is an example that will keep our query strings squeaky clean:

```
<IfModule mod_rewrite.c>
 RewriteEngine On
 RewriteCond %{QUERY_STRING} ^.*(localhost|loopback|127\.0\.0\.1).*                    [NC,OR]
 RewriteCond %{QUERY_STRING} ^.*(\.|\*|;|<|>|'|"|\)|%0A|%0D|%22|%27|%3C|%3E|%00).*          [NC,OR]
 RewriteCond %{QUERY_STRING} ^.*(md5|benchmark|union|select|insert|cast|set|declare|drop|update).* [NC]
 RewriteRule ^(.*)$ - [F,L]
</IfModule>
```

As you can see, here we are using the QUERY_STRING variable to check all query-string input against a list of prohibited alphanumeric characters strings. This strategy will deny access to any URL-request that includes a query-string containing localhost references, invalid punctuation, hexadecimal equivalents, and various SQL commands.

8. Blacklist via IP Address

Last but certainly not least, we can blacklist according to IP address. Blacklisting sites based on IP is probably the oldest method in the book and works great for denying site access to stalkers, scrapers, spammers, trolls, and many other types of troublesome morons.

The catch is that the method only works when the perpetrators are coming from the same location.

An easy way to bypass any IP blacklist is to simply use a different ISP or visit via proxy server.

Even so, there is no lack of mindless creeps out there roaming the Internet, who sit there, using the same machine, day after day, relentlessly harassing innocent websites. For these types of lazy, no-life losers, blacklisting via IP address is the perfect solution.

Here is a hypothetical example demonstrating several ways to blacklist IPs:

```
# block individual IPs
<IfModule mod_rewrite.c>
 RewriteEngine On
 RewriteCond %{REMOTE_ADDR} ^123\.456\.789\.1 [OR]
 RewriteCond %{REMOTE_ADDR} ^456\.789\.123\.2 [OR]
 RewriteCond %{REMOTE_ADDR} ^789\.123\.456\.3 [OR]
 RewriteRule ^(.*)$ - [F,L]
</IfModule>

# block ranges of IPs
<IfModule mod_rewrite.c>
 RewriteEngine On
 RewriteCond %{REMOTE_ADDR} ^123\. [OR]
 RewriteCond %{REMOTE_ADDR} ^456\.789\. [OR]
 RewriteCond %{REMOTE_ADDR} ^789\.123\.456\. [OR]
 RewriteRule ^(.*)$ - [F,L]
</IfModule>

# alt block IP method
<Limit GET POST PUT>
 order allow,deny
 allow from all
 deny from 123.
 deny from 123.456.
 deny from 123.456.789.0
</Limit>
```

1. In the first block, we are blacklisting three specific IP addresses using Apache's **mod_rewrite** and its associated **REMOTE_ADDR** variable.

2. Then, in the next code block, we are blocking three different ranges of IPs by omitting numerical data from the targeted IP string.

    - In the first line we target any IP beginning with "123.", which is an *enormous* number of addresses.
    - In the second line, we block a different, more restrictive range by including the second portion of the address.
    - Finally, in the third line, we block a different, much smaller range of IPs by including a third portion of the address.

3. In 3rd block, this is an equally effective method that enables you to block IP addresses and ranges as specifically as necessary. Each deny line pattern-matches according to the specified IP string.

Dealing with Blacklisted Visitors

In all blacklisting techniques, we respond to all blacklisted visitors with the server's default "**403 Forbidden**" error. This page serves its purpose and requires very little to deliver in terms of system resources, however there is much more that you can do with blacklisted traffic.

Here are a few ideas:

Redirect to home page

More subtle than the 403 error, this redirect strategy routes blocked traffic directly to the home page. To use, replace the RewriteRule directive (i.e., the entire line) with the following code:

```
RewriteRule ^(.*)$ http://yoursite.com/ [F,L]
```

Redirect to external site

The possibilities here are endless. Just make sure you think twice about the destination, as any scum that you redirect to another site will be seen as coming from your own. Even so, here is the code that you would use to replace the **RewriteRule** directive in any of the examples above:

```
RewriteRule ^(.*)$ http://anothersite.com/new.html [F,L]
```

Redirect them back to their own site

It's like having a magic shield that reflects attacks back at the attacker. Send a clear message by using this code as the RewriteRule directive in any of our blacklisting methods:

```
RewriteRule ^(.*)$ http://%{REMOTE_ADDR}/ [F,L]
```

Custom processing

This is the most useful approach for understanding your traffic and developing an optimal security strategy. The code would look something like this, depending on your file name and its location:

```
RewriteRule ^(.*)$ /home/path/blacklisting-script.php [F,L]
```