
ФАЗЗИНГ САЙТА «ЗАЩИЩЕННОГО» MOD_REWRITE

ТИМУР ЮНУСОВ, POSITIVE TECHNOLOGIES



POSITIVE / TECHNOLOGIES®

ОГЛАВЛЕНИЕ

1	ВСТУПИТЕЛЬНОЕ СЛОВО	3
2	ВВЕДЕНИЕ.....	4
3	МЕТОДИКА РАБОТЫ	7
4	ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ	12
5	ЗАКЛЮЧЕНИЕ.....	13
6	ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ:.....	14
7	ИСПОЛЬЗОВАННЫЕ МАТЕРИАЛЫ:	17

1 Вступительное слово

Все чаще в сети Интернет можно встретить сайты, которые скрывают передаваемые параметры приложению посредством модуля Apache – **mod_rewrite**. Часто у web-разработчиков создается иллюзия того, что таким образом можно защитить web-приложение от атак с использованием уязвимостей SQL Injection, Cross-Site Scripting и подобных. На самом деле, это является распространенным заблуждением, аналогичным тому заблуждению, что сокрытие *fingerprint* (отпечатков) сервисов увеличивает безопасность этих сервисов. Безусловно, использование **mod_rewrite** для сокрытия передаваемых параметров к приложению, ровно, как и сокрытие отпечатков является некоторой преградой для атакующего. Но, как говорится, «нет такой преграды, которую нельзя было бы обойти».

2 Введение

Mod_rewrite - достаточно мощный инструмент для URL-преобразований «на лету». Этот замечательный модуль web-сервера Apache предоставляет поистине безграничные возможности. На данный момент модуль чаще всего применяется для:

- поисковой оптимизации (SEO);
- защиты от прямых загрузок, путем сокрытия реального местоположения файла;
- сокрытия иерархии поступающих параметров, каталогов и сценариев web-приложения путем их централизованного динамического преобразования;
- разграничения доступа; **mod_rewrite** может проверять значения HTTP-заголовков, в т.ч. значение COOKIE на соответствие правилам и по результатам проверок проводить (или не проводить) перенаправление.

Чаще всего, настроенный **mod_rewrite** затрудняет возможности поиска и эксплуатации уязвимостей web-приложения. Рассмотрим причины этого:

1. Трудно распознать реальное предназначение элемента URL. Например, если мы видим ссылку вида *http://www.example.com/main/search/stroka_poiska*, невозможно понять, какой из элементов URL является **относительным путем** от корневого каталога web-сервера, что является **названием сценария**, а что является **параметром** этого сценария. Это сильно затрудняет анализ структуры web-приложения;

Например, следующие правила позволят одной и той же ссылке соответствовать различным реальным представлениям структуры web-приложения:

```
RewriteRule ^search/(.+) $ search.php?search=$1
```

При таком правиле ссылка

```
http://www.example.com/main/search/stroka_poiska
```

соответствует сценарию *search.php*, который располагается в папке */main/* относительно корневого каталога web-сервера, вызванному с параметром *search=stroka_poiska*

```
RewriteRule ^(.+)/(.+) $ script.php?act=$1&value=$2
```

По данному правилу та же самая ссылка соответствует сценарию *script.php*, находящегося в папке *main*, вызванному с параметрами *act=search&value=stroka_poiska*

```
RewriteRule ^(.+)/(.+)/(.+)$ $1.php?value1=$2&value2=$3
```

А по этому правилу такая же ссылка соответствует сценарию *main.php*, который находится в корневом каталоге web-сервера, вызванному с параметрами *value1=search&value2=stroka_poiska*

Кроме того, благодаря гибкости используемых **mod_rewrite** регулярных выражений, даже схожие по виду URL-адреса могут обращаться к абсолютно разным сценариям :

Например, данные правила:

```
RewriteRule ^(.+)/(.+)$ script1.php?value1=$2&value2=$3
```

```
RewriteRule ^(.+)/(.+)/(.+)$ script2.php?value1=$2&value2=$3
```

отправят 2 схожих запроса разным сценариям:

```
http://www.example.com/stroka1/stroka2
```

```
http://www.example.com/stroka1/stroka2/stroka3
```

2. Трудно определить язык программирования, на котором написано приложение. За ссылкой <http://www.example.com/main/articles/statya.html> может находиться как серверный сценарий на языке PHP или ASP или Perl, так и статическая страница на языке HTML;
3. Наличие регулярных выражений, которые можно использовать в правилах перезаписи **mod_rewrite**, позволяют фильтровать входные параметры (далее по тексту будет рассмотрен пример);
4. При автоматизации поиска уязвимостей необходимо заменять некоторые символы (например, слеш - / на %2F (hexadecimal encoding) или %252F (double encoding))[1] по причине их обработки еще на стадии «разбора» URL-адреса mod_rewrite-ом.

В связи с этим, многие разработчики[2][3] и администраторы предпочитают «маскировать» наличие уязвимостей с помощью **mod_rewrite**, а не обнаруживать



и исправлять проблемы. Но такой подход, как и любой метод, основанный на подходе «Security Through Obscurity» работает весьма плохо.

3 Методика работы

Основная идея поиска уязвимостей при включенном **mod_rewrite** – это перебор (*brute-force*), позволяющий определить реальные имена реальных параметров серверных сценариев, в которые подставляются значения из правил перезаписи (*rewrite rules*) **mod_rewrite**. Основные особенности перебора:

- использование в одном запросе множества параметров (количество параметров ограничивается максимальной длиной URL - (по умолчанию для web-сервера Apache 2.x – 8192 символа, для IIS – 16 384 символа);
- применение метода бинарного поиска (дихотомии)[4] для обнаружения необходимых параметров;
- словарный перебор (использование популярных названий параметров и префиксов) - *id*, *count*, и т.д. в сочетании с полным перебором названий параметров (комбинированные атаки);
- различные варианты анализа результатов;
- возможность рекурсивного поиска параметров (для поиска всего множества параметров одного сценария).

Рассмотрим алгоритм поиска уязвимостей в web-приложении при включенном **mod_rewrite**.

1. Определение реального имени серверного сценария.

Используются стандартные и популярные названия сценариев, такие как *index.php*, *main.php*. Необходимо определить, существует ли данный сценарий или нет (например, по наличию ошибки «**404 – Not Found**»). Иногда правила могут перезаписывать любые URL-адреса, начиная с корневого каталога web-сервера, то есть, запрос *http://www.example.com/index.php* уже будет вести на абсолютно другой сценарий (*RewriteRule ^(.+)\$ script.php?\$1*). В таком случае, дальнейшие проверки не имеют смысла.

Примечание:

После публикации статьи на английском языке в блоге <http://ptresearch.blogspot.com> один венгерский *интересующийся* (некто [0x32353031](#)) предложил свой способ раскрытия имени реального серверного сценария: с помощью принудительного вызова ошибки веб-сервера *413 "Request Entity Too Large"*. Указанная ошибка возникает в случае, когда веб-сервер отказывается обработать запрос по причине слишком большого размера тела запроса. Состояние *413 HTTP* может быть вызвано принудительно путем установки заведомо некорректного размера длины запроса, например: **"Content-length: x"**. К сожалению, как показали эксперименты, проведенные [Дмитрием Евтеевым](#), этот метод работает исключительно под Apache 2.x.

2. Определение параметров поступающих в приложение

Наиболее часто разработчиками web-приложений используются названия параметров вроде *id*, *file* и т.д. Учитывая это, необходимо проверить:

- самые популярные названия переменных (*id*, *path*, *page*, *debug*, *cat* и проч.) – *словарь популярных названий*;
 - короткие названия переменных (1-5 символов) в алфавите [a-z0-9-_] – полный перебор;
 - «гибридные» названия – по формулам:
 - «префикс» + «популярное название параметра»;
 - «популярное название параметра» + «постфикс»;
 - «полный перебор» + «разделитель (_, -)» + «популярное название параметра»;
 - «популярное название параметра» + «разделитель (_, -)» + «полный перебор».
- для кода, в названиях переменных которого используются различные суффиксы и префиксы;
- переменные-массивы (`param[]`)[5].

О последнем типе параметров нужно уточнить: дело в том, что если в PHP-сценарии к параметру, инициализированному как массив (`http://example.com/index.php?param[]=value`), произойдет обращение как к простому типу, будет выведена ошибка (в зависимости от *error_reporting level*), что приведет к раскрытию установочного пути.

Так же можно использовать:

- массивы `GLOBALS` (`http://example.com/index.php?GLOBALS[var]=value`)[6][7][8];
- стандартные переменные `_SERVER` (в определенных случаях можно перезаписать и их [7][8]);
- переменные в сочетании с их *zend_hash_key* (для обхода уязвимых функций `unset()` [9]).

3. Определение значение параметров

Для решения этой задачи важно использовать различные варианты параметров. Ведь, если, например, везде подставлять в значение =1, то велика вероятность, что оно совпадет со значением «по умолчанию» переменной и сервер вернет тот же ответ. Также различные параметры - это различные ошибки. А если проводить поиск по значению параметра в ответе (потенциальные **XSS, Local File Including, Path Traversal**, etc), то для каждого параметра потребуется генерировать уникальное значение.

Возможные варианты, их предназначение, плюсы и минусы:

- Числовое значение: 0,1,2,... Самый простой вариант. Можно обойтись 3 вариантами – 0,1 и больше 1. Из плюсов – минимальная длина строки запроса, что минимально сказывается на производительности работы.
- Ошибочное значение: «\», «./», «a%00» и т.д. – различные наборы символов, которые могут привести к потенциальным ошибкам, по сигнатурам которых можно определить присутствие параметра.
- Фиксированное значение параметра. Например, если существует URL `http://example.com/main/search/stroka_poiska`, то, зафиксировав ответ сценария на значение `stroka_poiska` и подставив это значение во все параметры, то по ответу, совпадающему с эталонным значением, можно будет найти параметр, отвечающий за конкретную позицию в URL-адресе.*
- Случайное число. Генерируя достаточно длинные случайные числа (5-9 символов), можно искать совпадение с этими числами в ответах сценариев. Случайные числа дают большую вероятность отсутствия ошибки второго рода (*False Positive*)

* - например, URL `http://example.com/main/search/test` отвечает за сценарий поиска с параметром «test». Нам необходимо найти оригинальный параметр строки поиска. Для этого мы фиксируем сигнатуру со страницы `http://example.com/main/search/test` (например, «**по запросу test найдено**») и проводим перебор по всем параметрам, подставляя в значение параметра «test». Анализируя ответы по наличию сигнатуры, можно найти необходимый параметр.

4. Составление запроса и подбор

Составить запрос вида

`http://example.com/script.php?param1=value¶m2=value&...&abc=value.`

Ограничение на длину URL запроса составляет 8192 символа (для сервера Apache). Это означает, что все параметры из алфавита [a-z0-9] длины до 4 символов переберутся примерно за 5880 запросов. При хорошей скорости интернета, это потребует 3-5 минут.

5. Анализ ответов и определение наличия параметра

Эта часть работы алгоритма самая важная, поскольку именно она определяет эффективность работы. Различия в структуре web-приложений и различные условия обуславливают решение о выборе того или иного метода анализа ответа. Рассмотрим плюсы и минусы различных подходов:

- определение по длине ответа:

ПЛЮСЫ:

- Самый быстрый и простой способ. Необходимо лишь узнать длину эталонного запроса (без параметров) и сравнивать ее с длиной запросов с параметрами.

МИНУСЫ:

- Данный метод неприменим в ситуациях, когда каждый раз сценарий генерирует ответ с уникальным содержимым (баннеры, случайный контент, наличие строки запроса в ответе).

- определение по сигнатурам случайных значений переменных:

ПЛЮСЫ:

- Уменьшение числа ложных срабатываний.

МИНУСЫ:

- Необходимо удалять ложные срабатывания, связанные с штатным попаданием случайного значения в ответ (например, при штатном наличии данных запроса в ответе).
- Уменьшение скорости из-за увеличения длины значений параметров (с 1-2 символов до 5-7 символов, что увеличивает время перебора в 1.5-2 раза для параметров длиной до 4 символов).

- определение по фиксированным ответам*

ПЛЮСЫ:

- точность поиска.

МИНУСЫ:

- точечность поиска – поиск происходит только по одному конкретному параметру.

** - например, когда необходимо определить параметры на странице http://example.com/main/search/stroka_poiska, нам необходимо подставлять во все переменные значение «stroka_poiska» и отслеживать ответ. Верным будет параметр, попавший в запрос, ответ на который совпал по содержимому с первоначальной страницей.*

- определение по сигнатурам ошибок

ПЛЮСЫ:

- почти полное отсутствие ошибок второго рода (False Positive – ложное срабатывание), что исключает идентификацию наличия параметра в запросе, когда на самом деле его в нем нет.

МИНУСЫ:

- требуется использовать базу сигнатур ошибок;
- если наличие параметра не вызывает ошибку, то данный метод не выявит параметр.

Как видно из вышперечисленного, различные методы эффективны в разных ситуациях. Самым простым и, порой, наиболее действенным является первый подход, но иногда нельзя выявить параметр даже с сложных методов. Поэтому желательно использовать различные методики или определяться в каждом конкретном случае, какая методика уменьшит число ложных срабатываний. В конце-концов, если в результате работы будет выявлено 10-15 переменных, они всегда могут быть проверены на предмет ложных срабатываний вручную.

4 Практическая реализация

В качестве практической демонстрации эффективности приведенных методов может быть использована утилита, доступная для загрузки по следующему адресу: http://www.ptsecurity.ru/download/modrewrite_search.zip.

Утилита реализует следующие возможности:

1. Полный перебор по алфавиту, наиболее часто встречаемым параметрам (указываются в файле `params.txt`), в комбинированном режиме («популярное название параметра» + «разделитель (`_`, `-`)» + «полный перебор»). Так же можно указать флаг, при котором каждая переменная будет представлена в виде массива (`param[]`).
2. Использование дихотомии для поиска параметров.
3. Поиск по длине ответа (указывается первоначальная страница, ее размер)
4. Выбор первоначальной страницы с параметрами (например, `http://example.com/index.php?page=admin`), для поиска множества вложенных параметров (указанные параметры, естественно, будут исключаться из проверки)
5. Выбор символов, подставляемых в значения переменных.

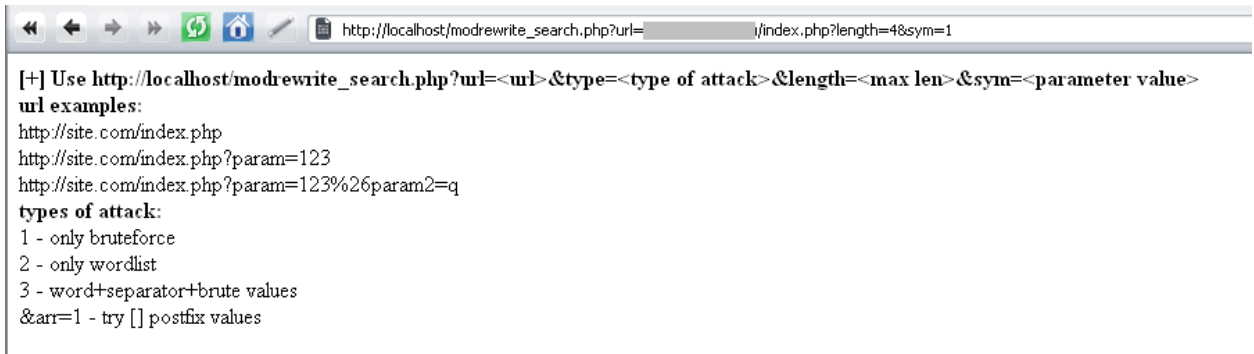
Некоторые из этих механизмов реализованы в системе контроля защищенности и соответствия стандартам MaxPatrol.

5 Заключение

Приведенный метод подходит не только для оценки защищенности web-приложений, использующих **mod_rewrite**, но и для поиска переменных, которые могут быть перезаписаны при включенном параметре *register_globals*. а так же для поиска недокументированных возможностей, таких как различные отладочные («*debug*») режимы и т.д.

6 Примеры работы программы:


Параметры, необходимые для работы:



```
http://localhost/modrewrite_search.php?url=.../index.php?length=4&sym=1

[+] Use http://localhost/modrewrite_search.php?url=<url>&type=<type of attack>&length=<max len>&sym=<parameter value>
url examples:
http://site.com/index.php
http://site.com/index.php?param=123
http://site.com/index.php?param=123%26param2=q
types of attack:
1 - only bruteforce
2 - only wordlist
3 - word+separator+brute values
&arr=1 - try [] postfix values
```

Полный перебор до 3 символов:



```
http://localhost/modrewrite_search.php?url=http://.../index.php&length=3&sym=1&type=1

[+] Analyze http://i...ru/index.php?
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find! http://i...l.ru/index.php?id=1.&
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find! http://...l.ru/index.php?say=1.&
[+] Done
```

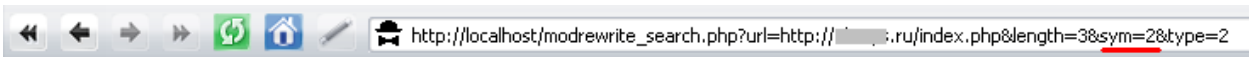
Перебор по словарю уже другого ресурса:



```
[+] Analyze http://\[redacted\].ru/index.php?
[+] Wordlist loaded...36 words.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find! http://\[redacted\].ru/index.php?deb=1&
[+] Done
```

(параметр length тут не имеет значения)

Обратите внимание, что «пойман» еще один параметр при использовании другого значения параметров:



```
[+] Analyze http://\[redacted\].ru/index.php?
[+] Wordlist loaded...36 words.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find! http://\[redacted\].ru/index.php?deb=2&
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find! http://\[redacted\].ru/index.php?page=2&
[+] Done
```

Отсюда следует, что index.php без параметров соответствует index.php?page=1

А вот так можно выявить комбинированной атакой параметр, который сложно было бы выявить полным перебором и которого нет в словаре:



```
http://localhost/modrewrite_search.php?url=http://[redacted]/index.php&length=2&sym=26&type=3

[+] Analyze http://[redacted]/index.php?
[+] Wordlist loaded...37 words.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find smth Try to detect.
[+] Find! http://[redacted]/index.php?act_id=26&
```

7 Использованные материалы:

1. http://www.owasp.org/index.php/Double_Encoding
2. <http://dimoning.ru/kak-napisat-svoy-dvizhok-bloga-1.html>
3. <http://webscript.ru/stories/07/02/01/2099269>
4. http://ru.wikipedia.org/wiki/Двоичный_поиск
5. <http://raz0r.name/mysli/proveryajte-tip-dannyx/>
6. <http://www.hardened-php.net/globals-problem>
7. http://www.hardened-php.net/advisory_192005.78.html
8. <http://www.wisec.it/vulns.php?id=10>
9. http://www.hardened-php.net/hphp/zend_hash_del_key_or_index_vulnerability.html