

Abusing poor programming techniques in webserver scripts V 1.0

Source: <http://www.derkeiler.com/Mailing-Lists/securityfocus/secprog/2001-07/0001.html>

From: memonix@roses-labs.com

Date: 07/26/01

From: memonix@roses-labs.com

To: secprog@securityfocus.com

Date: Thu, 26 Jul 2001 12:42:47 -0600

Subject: Abusing poor programming techniques in webserver scripts V 1.0

Message-ID: <3B601047.32657.149C14E@localhost>

Roses Labs
Advanced Security Research
<http://www.roses-labs.com>

Title :

Abusing poor programming techniques in webserver scripts V 1.0

Authors:

Memonix <memonix@roses-labs.com>

MrJade <mrjade@roses-labs.com>

RL CODE :

RLD003en

Date :

23/7/2k1.

Neither the authors or Roses Labs accepts any responsibility for the damages
this paper may cause.

Following our Confidential Policy we have not published any vulnerable URL.

However, these poor coding techniques affects hundreds of websites.

Spanish Speakers / Hispano Hablantes.

Disponibile una version en español de este documento en el website de Roses Labs.

---== Introduction ==-----

SecurityFocus SecProg: Abusing poor programming techniques in w

This technique does not show a failure in database managers, but an open front door in the security of authentication systems. Its implementation is only possible in databases queries with a certain structure, although it is possible to go more deep in the technical test and error within the verification routine.

This paper isn't oriented to ANY SCRIPT IN CONCRETE :), it'll focus the problem as generic as it can be done.

While this code is used in logins scripts:

```
$login = Request.Form("login")  
$password = Request.Form("password")
```

```
SELECT field FROM database WHERE Login=$login AND Password=$password
```

It is possible to validate a login without knowing both the user name or the password.

Certain programming norms must moreover be fulfilled so that this technique works, but they are very basic norms and in 80% of the cases they are almost not considered.

1. Select statement must be like this one or similar.

```
SELECT field FROM database WHERE Login=$login AND Password=$Password
```

Where \$login and \$password are form fields that the user must fill with it's own data.

2. Server scripts must NOT check both \$login and \$password for bad characters in variables, as ' ;) " # | and all those characters will fool the database parser. The existence of invalid characters is not verified in the script of access to the database, not in the html using javascript, as it can be defeated too. The confirmation usually is made in the same html form using javascript, but this can be avoided adding the fields from the form to the url as if they were parameters of a GET petition and to carry out the shipment without the confirmation of the javascript, or using a direct connection with http server with a telnet client. To avoid a fake login, it should be check in the script file on the server that pick up the request data.

3. Once validated, no checks must be done on \$login and \$password variables as they will return blank strings or incorrect strings formats. The SQL statement will be parsed by the database manager, but couldn't be understood by cgi script on server, and it will return script error.

With all these conditions in the server it's posible to exploit this bug on ANY login script.

---== Abstract ==-----

This is a "poor" code example:

```
SELECT field FROM database WHERE Login=$login AND Password=$Password
```

Where \$login and \$password are form fields that user must fill with it's own data.

```
SELECT field FROM database WHERE Login="" AND Password=""
```

When no input is inserted in the fields. Those "", are inserted by the script parser and couldn't be avoided, so it's limiting the vulnerability to strings that will terminate ALL the "" on statment.

A user login will be parsed as:

```
SELECT field FROM database WHERE Login='Jon' AND Password='1234'
```

When login = john and password is 1234, SQL operators may be inserted too in the form fields so if you insert as login the character ' and it's not checked before it's parsed, will return a SQL Bad command error from SQL database manager.

```
SELECT field FROM database WHERE Login="" AND Password=""
```

```
Unknown DB Provider for ODBC Drivers error '80040e14'  
[Unknown][Driver ODBC data Access] Sintax error in query expression  
'UserName="" AND PassWord="';'  
/script/loginscript.asp, line xx
```

If we take care of all the characters involved in a SQL statment it's possible to fake the query. For example

if we insert as
login ' or ''=
password ' or ''=

```
SELECT field FROM database WHERE Login=$login AND Password=$Password
```

Will be parsed as

```
SELECT field FROM database WHERE Login="" or ""="" AND Password="" or ""=""
```

and we can seen that, or ""="" will allways return TRUE.

What will happen then?

The SQL statement is going to be check in the database. The first input is checked with the SQL query and returns TRUE, the program (server

SecurityFocus SecProg: Abusing poor programming techniques in w

script) will continue as if a valid login is completed. So for the server script, if no more CHECKS are done, the user logged is the first user on the database, and usually, the first user in the database is the database Admin added to it and as it's started with A letter, and the database it may be sorted by login.

But, the program flow may check now login or password fields, and if it happen, may fail as those strings are incorrect for the server script language, and will return an SCRIPT ERROR from server, script server, not the database server.

---== Getting into ==-----

As explained below, there are a lot of ways to skip the login validation if all previous conditions are presented in the login script. All of them are SQL based and has been checked within the development of Dyl.

Any one that match an SQL statement can be used for gaining access to those unsecure scripts. For example:

It's possible to gain access as a known user only knowing it's login or password. Only one of them is needed.

To force a login knowing only the username the login SQL statement must be filled avoiding password check as follows:

```
login Jonn' and "=" or "="  
password ' or "=" or anyone you want to insert.
```

Once parsed, 'Jonn' and "=" or "=" will return TRUE when user Jonn on the database is checked against the SQL statement.

Also, knowing the password, a login may be forced too:

```
login ' or "=" or anyone you want to insert.  
password 1234' and "=" or "="
```

So when the password is checked in the SQL statment it will return TRUE if it is not found a mattering username or login.

There's another vulnerability more funny than this one and it's using SQL comment characters.

```
login : /*  
password: */ OR " = "
```

```
SELECT field FROM database WHERE Login = '/*' AND Password = '*/ OR " = "
```

SQL statment within /* */ characters is avoided and SQL may be modified to easily gain access, as it's transformed into:

```
SELECT FROM DATABASE WHERE Login = " OR " = " ALWAYS TRUE
```

There are some other things involved in SQL comments on inputs.

```
login: ' /*plain text */ or "= '  
password: ' /* more plain text */ or "='
```

This give us the possibility of inserting additional data on the input that avoid the database manager parser allowing us to insert text for other purposes. For example.

If the email address is checked as the login name, it's possible to fake a login that complains both SQL statment and the script email validation as it follows:

```
login: '/*mi@mail.com  
password: */ or "='
```

The server script will parse the \$login variable for a valid email such *@*.*, and SQL will parse it avoiding comments (/* */) allowing a login.

And so on.. different combinations for different scripts..

--== Resolving this ==-----

There are several ways to drop those fake logins, some are javascript base, but those checks may be skipped using arguments as a GET command and adding in the form fields variables as ?Login=' or " ='&password=' or "=', or using a telnet client.

Although it can be included as a fast patch in the login script. This one can be an example:

Also the login form must be modified too, so before the submit process you must check both username and password.

Here we can see two server scripts codes from Bat[e].

BEFORE:

```
if ($luser!="" && $lpassword!="") {  
$login_rs = @mysql_query("SELECT * FROM cro_user WHERE  
user_login='$luser' AND user_password='$lpassword' AND user_status!='0'  
LIMIT 0,1",$db);  
if (@mysql_num_rows($login_rs)==0) {  
    echo "LOGGED ON";  
    }  
}
```

SecurityFocus SecProg: Abusing poor programming techniques in w

This code presents a bug that can validate login using as login and pass:

```
login : '|  
password: '|
```

The code has been fixed as follows.

AFTER:

```
$luser = trim(htmlspecialchars(addslashes($luser)));  
$lpassword = trim(htmlspecialchars(addslashes($lpassword)));  
  
if ($luser!="" && $lpassword!="") {  
$login_rs = @mysql_query("SELECT * FROM cro_user WHERE  
user_login='$luser' AND user_password='$lpassword' AND user_status!='0'  
LIMIT 0,1",$db);  
if (@mysql_num_rows($login_rs)==0) {  
    echo "LOGGED ON";  
    }  
}
```

All the special characters in the login strings will be dropped in the SQL. The main objective is to delete or drop any tries of login when it includes these dangerous characters:

() / , ; . : # < > | \ " All but letters and numbers.

We recommend, that all the checks must be done in the server script before the SQL parsing.

==== Detecting an Attack ==-----

The database agent doesn't log a bad entry in SQL statment. That means, that a tool for examining these kind of bug in a login script can brute force the web server. Only the http server logs can be examined for any attack but can't be considered as certain kind of attack. Some vendor we have contacted told us they use their own scripts for logging it, and now they have added new line saving the login variables for examining them.

Other traffic analyzers as Snort may be programmed to detect those bad characters in login fields to notice the web master that something is happening, but it can also be in serious troubles detecting :// string in http:// petitions.

A notice must be included too in all login scripts examples as it can be, avoiding a new programmer for doing it's first login script insecure. We are contacting any ASP, CGI, PERL or PHP web resource where a bug has been found, helping to fix it as well as we can.

==== Bad SQL ==-----

- *Messages sorted by:* [date] [thread] [subject] [author] [attachment]