Jackson State University
Department of Computer Science
CSC 438/539 Systems and Software Security, Spring 2014

Instructor: Dr. Natarajan Meghanathan

Project 2: Simulating the TOCTTOU Vulnerability in a Linux-Java Environment

Due Date: March 26, 2014, 7.30 PM Max. Points: 100

Project Objective: The objective of this project is to demonstrate the TOCTTOU (Time of Check to Time of Use) vulnerability in Linux environment using a simple file updater program written in Java.

Introduction to TOCTTOU Vulnerability

TOCTTOU stands for Time of Control to Time of Use. This type of vulnerability occurs when a user is granted permission to access to a resource or file at a particular time and if access to the file or resource is taken away from the user, the user may still access the file or resource if the user has never relinquished control over the asset.

In this activity, you will simulate this kind of vulnerability using a text file in an Ubuntu virtual machine. You will create two users in the operating system: User A and User B (see the details below for the exact naming convention for the user names). User A will create a text file and grant permission to everyone to read and write to it. User B will then access this file via a simple program (written in Java). While User B is accessing the file, User A will revoke User B's permission to read from and write to the file. Even though User B loses permission to access the file, User B has *already* been granted permission to access the file, and is able to continue reading and writing to the file until it is released.

It is important that you read this entire document first, as some of the steps are time-sensitive. You will need to know in advance what you will need to do in order for the activity to work properly.

What to Submit (Submit the recorded video through GoogleDrive using your JSU email address; after uploading the video, e-mail the link to natarajan.meghanathan@jsums.edu):

After you setup the VM, install Java JDK, type and compile your program in user B's account, start the recording and record as follows:

(1) Login to User A's account; create a text file (owned by user A) and set its permissions such that others can read and write to it.
(2) Login to User B's account in another terminal. Launch the file updater program to write to the text file of user A.
(3) As it is running, go to the terminal where you have logged in as User A and displays the contents of the text file. The text file's contents should reflect the updates from user B's file updater program. Now, let user A to change the permission for others to be not able to write to the file (the read permission is still granted).
(4) Return to the terminal for User B and show that the program is still running without any problem/error reported. After a couple of minutes, go the terminal for user A and display the contents of the text file. Explain the TOCTTOU vulnerability that is behind the fact that the contents of the file continuing to be updated.
(5) After the file updater program at user B stops, launch it again. Observe what happens when you try to start fresh and run the program at user B. Explain the reason behind what you observe.

Desktop Video Recording

You could try using one of the **desktop recording software** (or anything of your choice):
CamStudio: http://sourceforge.net/projects/camstudio/files/legacy/
Debut: http://www.nchsoftware.com/capture/index.html

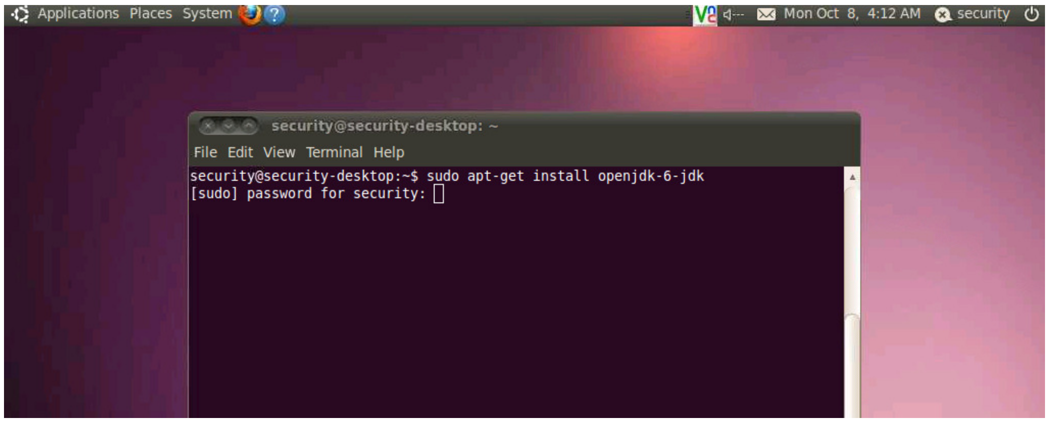
Creating the Programming Environment: Ubuntu/Linux VM

Task 1:

You will use the Ubuntu VM that you installed in Oracle Virtualbox or VM Player for Project 1. If you have not already installed Ubuntu VM in Oracle Virtualbox, read the description for Project 1 to complete this task.
To install Java on the Ubuntu VM, follow the instructions in Task 2.

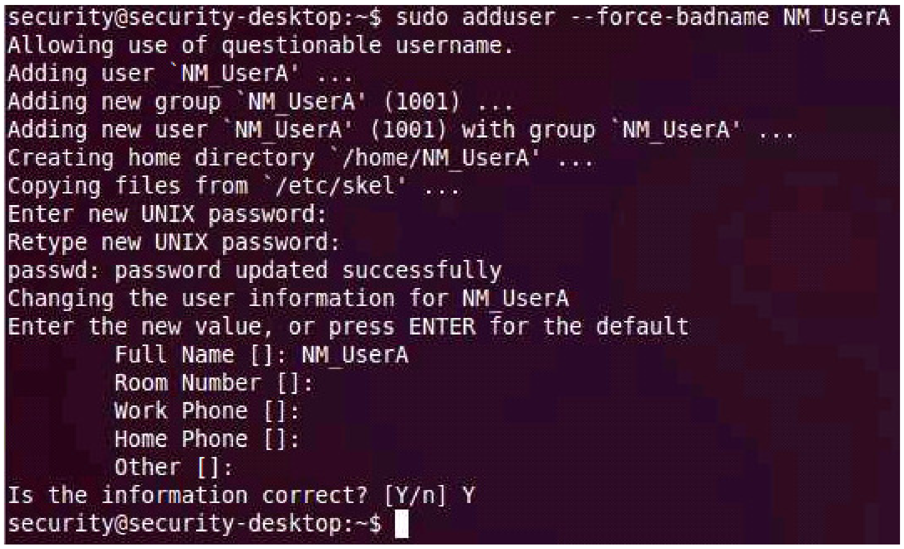
Task 2: Installing Java on your VM

Open a terminal by using Applications --> Accessories --> Terminal. Type "sudo apt-get install openjdk-6-jdk" and press enter. You will then be prompted for your password. This will install the Java Development Kit.


Applications Places System
Mon Oct 8, 4:12 AM
security
security@security-desktop: ~
File Edit View Terminal Help

```
security@security-desktop:~$ sudo apt-get install openjdk-6-jdk
[sudo] password for security: 
```

Task 3: Creating the Two User Accounts



```
security@security-desktop:~$ sudo adduser --force-badname NM_UserA
Allowing use of questionable username.
Adding user `NM_UserA' ...
Adding new group `NM_UserA' (1001) ...
Adding new user `NM_UserA' (1001) with group `NM_UserA' ...
Creating home directory `/home/NM_UserA' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for NM_UserA
Enter the new value, or press ENTER for the default
        Full Name []: NM_UserA
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] Y
security@security-desktop:~$ 
```

[NM_UserA, as above] and [NM_UserB, when the procedure is repeated]

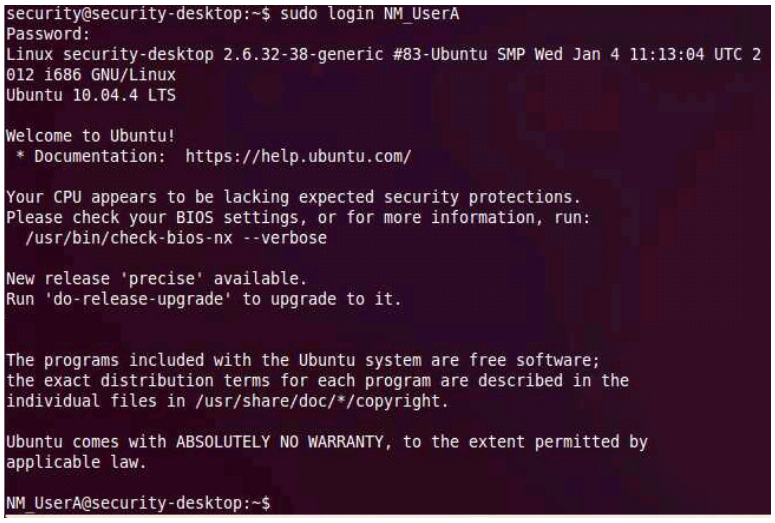
Make sure you install Java on your VM, before you create the two user accounts and logging in as the two users.
Open a terminal by using Applications --> Accessories --> Terminal. Let the two user accounts you create be named using the following convention.. [FirstInitial][LastInitial]_UserA and [FirstInitial][LastInitial]_UserB

In my case, I am going to create the two user accounts with name NM_UserA and NM_UserB.

You would have to use the "--Force-badname" option to let the system to create the usernames of your choice. You will create a user account as shown in Figure 1.

Following the above approach, you can create user accounts NM_UserA and NM_UserB. Make sure to remember the usernames and passwords you setup for the two user accounts that you create.

Task 4: User A: Creating the text file to update



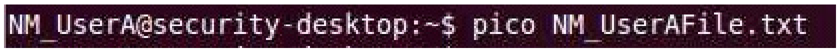
```
security@security-desktop:~$ sudo login NM_UserA
Password:
Linux security-desktop 2.6.32-38-generic #83-Ubuntu SMP Wed Jan 4 11:13:04 UTC 2
012 i686 GNU/Linux
Ubuntu 10.04.4 LTS

Welcome to Ubuntu!
 * Documentation:  https://help.ubuntu.com/

Your CPU appears to be lacking expected security protections.
Please check your BIOS settings, or for more information, run:
  /usr/bin/check-bios-nx --verbose

New release 'precise' available.
Run 'do-release-upgrade' to upgrade to it.


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

NM_UserA@security-desktop:~$
```

You will now login as User A using the following sudo command:

sudo login NM_UserA

Inside the NM_UserA account, create a text file by name NM_UserAFile.txt, depending on the naming convention that corresponds to you.

Use an editor of your choice to create the text file. I use the pico editor to create the text file. In the pico editor, after typing the contents, I press Ctrl+X to save and exit. The contents of the text file should be your name and the name of the course.



```
NM_UserA@security-desktop:~$ pico NM_UserAFile.txt
```

After you type the contents of the file, save and exit; set the permissions for **others** to be able to "read and write" to the file. Note that User A considers User B to belong to the 'others' category.

Then, use the *ls -l* command to display the permissions of the files in the folder.

Task 5: User B: Creating the Java program to update the text file

Open another terminal. You will now login as User B using the following sudo command:

```
sudo login NM_UserB
```

Again, using an editor of your choice, type the Java program to update the text file you created in user A's account. The following description will guide you through the development of the Java program. This java program will do several things. First, it will open _UserA's text file, read the contents and print them out to the screen. Then, it will open the text file for writing. It will initiate a ten-minute loop in which a new line is written to the file every minute. Once the ten minutes has passed, the file will be closed, and the program will end.

When dealing with files, you will need to import the java.io package. You will also need to import the java.util package, since we will be using the Date class later. The syntax for importing a package is

```
	import java.io.*;
```

You will need to repeat this process to import the java.util package.

Once you have imported the necessary packages, begin your Java class with

```
	public class Meghanathan_fileUpdater
	{

	}//end class Meghanathan_fileUpdater
```

where Meghanathan is my last name and the name of the file. The name of the class and the name of the file will always need to be the same. The // denotes the beginning of a comment.

Inside the class declaration, you will need to insert your main() method. This syntax for a main method is

```
	public static void main(String args[])
	{

	}//end method main()
```

Inside of the main() method, you will need to insert a try-catch block. This is necessary when using objects and methods which are likely to cause an error to be generated. Since there are a number of exceptions that can occur during file i/o, file operations always need to be included inside of a try block.

The syntax for a try-catch block is

```
	try
	{
		//any code that might throw an exception
	}//end try
	catch (Exception e)
	{
		//the code to execute when an exception is caught
	}//end catch()
```

If there is more than one type of exception that might be thrown, more specific exceptions can be caught through the use of multiple catch blocks. For example, working with files can cause either a FileNotFoundException or a NullPointerException to be thrown. If we wanted to do something different whenever one of the exceptions occurred, we could use several catch blocks, as below:

```
   try
   {
         …
   }//end try

catch (FileNotFoundException  f)
{
   //code to handle the FileNotFoundException
}//end catch(FileNotFoundException)

catch (NullPointerException  n)
{
   //code to handle the NullPointerException
}//end catch(NullPointerException)
```

Each type of exception that can be thrown is an object in java. Each exception object contains a method printStackTrace() which can be used to print out the order in which the error propagated through the call stack. This is particularly useful in debugging, since it will tell you exactly what kind of error occurred and on what lines in the code it occurred.

The main outline of the program should now be constructed. It will appear as follows:

```
import java.io.*;
import java.util.*;

public class Meghanathan_fileUpdater
{
   public static void main(String  args[])
   {
         try
         {

         }//end try
         catch (Exception  e)
         {
               e.printStackTrace();
         }//end catch
   }//end method main()
}//end class Meghanathan_fileUpdater
```

5.1 Open a file

The first thing that the program needs to do is open the file for reading. This will require the creation of three objects, a File object, a FileInputStream object, and a BufferedReader object. In the description below, the [*x*] means *x* is the name of the object. The file object represents the file that we want to open. In this case, it can be created as:

File [File object name] = new File("/home/[_UserA's complete username]/[file name].txt");

The FileInputStream object will take data from the file and insert it into a stream. The FileInputStream constructor takes the name of a File object as its argument:

FileInputStream [FileInputStream object name] = new FileInputStream([File object name]);

The BufferedReader removes data from an input stream and places it in a buffer where it may be used. The code for the creation of a BufferedReader is as follows

BufferedReader [BufferedReader name] = new BufferedReader(new InputStreamReader([FileInputStream name]));

5.2 Read from the file

Now that a BufferedReader has been created, its readLine() method may be used to read a line at a time from the file. Each line should be stored in a String variable, in this case, *line* :

String line = null;

We can use a while loop to repeatedly retrieve lines from the BufferedReader as long as more lines exist. In the while() loop below, the loop continues until the BufferedReader reads a null line, indicating that the end of the file has been reached.

```
while ( ( line = [BufferedReader name].readLine() ) != null)
{
   System.out.println(line);
}//end while
```

Once all the data has been read from the file, it needs to be closed. This is done by calling the FileInputStream's object's close() method.

5.3 Write to the file

Now that we have read the contents of the file, we will open it so that we may
append it. Appending a file means to open it and write to the end of it. This way, none of the file's contents are overwritten. This is done with the creation of a FileWriter object. The FileWriter class contains a constructor which takes two parameters: the name of File object, and a Boolean value that determines whether the file is to be opened for appending. We already have our File object, and we would like to append to the file, so our FileWriter declaration will look like:

FileWriter [FileWriter name] = new FileWriter([File object name], true);

5.4 Print the time

Now that the FileWriter object has been created, you will need to set up a loop. This can be a for() loop, which executes through ten cycles. Inside of the for() loop, you should print the current time to the file. In order to get the current time, you can create an object of class Date. The Date class's toString() method will allow you to get the current time and date.

One way to do this is with code such as:

```
Date currentdate = new Date();
String currenttime = currentdate.toString();
```

You will then need to create a string which says something to the effect of "File edited at (currenttime)." It will make the file more readable to preface each instance of the string with two new-line escape sequences (\n). This will ensure that each listing of the current time occurs on its own line. Print this string out to the screen using System.out.

Once the complete string is created with two new-line characters, the message, and the current time, the string can be written to the file. This is done with the FileWriter object's write() method. This method takes three arguments: a string, the offset, and the string's length. For this program, the string is the message string you created, the offset is 0, and the length can be determined by calling the String object's length() method (stringobject.length()). A complete call to the write() method will look like (assume *out* is the name of the FileWriter object):

out.write(output, 0, output.length());

where *out* is the name of the FileWriter object, and *output* is the name of the output String object.

The write() method will need to be followed by a call to the flush() method. The flush() method clears the contents of the FileWriter's buffer and places it in the file. The syntax of this method call would be:

out.flush();

5.5 Wait

The loop will now need to sleep for **45 seconds** before executing again. This is easily done with the Thread.sleep() method. The sleep() method takes one parameter: an integer number of milliseconds for which to sleep. For example, if we wanted the program to stop for thirty seconds, we would use

Thread.sleep(30000);

One important thing to keep in mind is that this method does not keep time very accurately. For the purposes of this program, the time-keeping will be fine, but whenever you need precision time-keeping, this method will not work well.

Now that you have the contents of your for() loop complete, you can close the for() loop. Once you close the loop, you will need to close the output file. This is done with the following syntax:

out.close();

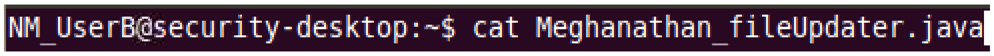
Once the file has been closed, insert a statement that will print out a message to the screen notifying the user that the program has finished executing.

Task 6: Run User B's File Updater Java Program

Now you have written the complete java program which opens a file for reading, reads and prints out its contents, and closes the file. The program then opens the file for appending (writing to the end), writes the current time, waits for one minute, and loops ten times (writing the current time to the file once a minute for ten minutes), and closes the file.

To compile the file, type *javac* followed by (a blank space and) the name of your .java file (including the .java extension), and press enter. If you are presented with any errors, you will need to find and correct them.

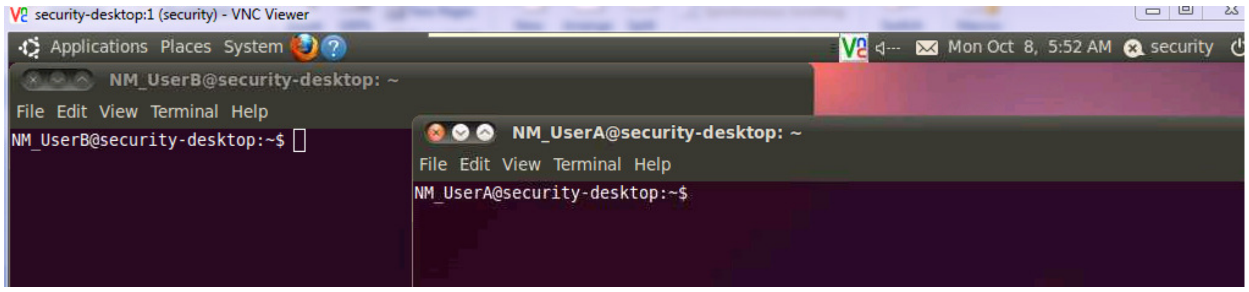
Once the file has compiled without any errors, use the *cat* command to print the contents of the complete the file updater Java program that you had just developed.



```
NM_UserB@security-desktop:~$ cat Meghanathan_fileUpdater.java
```

Once the file is compiled, you will need to run the file, but before you do, make sure that you know how to change file permissions (review Project 1, if needed). Once you begin running the file, you will have 450 seconds in which to change the file's permissions in order to get the full effect of this activity.

To run the file, type *java* followed by (a blank space and) the name of your java class (excluding the .java extension), and press enter. You will see the contents of the file output to the screen, and then you will not see any activity in the terminal window until the program is complete.


security-desktop:1 (security) - VNC Viewer
Applications Places System
Mon Oct 8, 5:52 AM
security
NM_UserB@security-desktop: ~
File Edit View Terminal Help
NM_UserA@security-desktop: ~
File Edit View Terminal Help

```
NM_UserB@security-desktop:~$
```

```
NM_UserA@security-desktop:~$
```

Task 7: User A: Changing the Permissions for User B

While the file updater Java program is running at the terminal for User B, go to terminal that you have opened for User A; wait for a minute or so, and then run the *cat* command to display the contents of the NM_UserAFile.txt file. It should display at least a couple of time lines.

Now, use the *chmod* command to remove the write permissions for **others**.

Use the *ls -l* command to display the permissions of the files in the folder for User A.

Let the Java program to complete running in the terminal for User B.

In the terminal for User A, use the cat command to display the contents of the NM_UserAFile.txt file.

Task 8: User B Running the File Updater Java Program (after User A removed the write permissions to the text file)

Now, in the terminal for User B, run the file updater Java program again. You should see an error message displaying the "Permission Denied" exception.

Explain why you now got an error message now and not before in Task 6.