Lab 1: Resource Allocation, Confidentiality, Injection

**Download the Kali VM for VirtualBox from:**

**O:\Information & Communications Technologies\Data\IT Program\ITSD ITSC315**

**or**

**USB stick during the first lab period.**

**Run the VM. Log in with: root/toor.**

# Problem 1: Heap Memory Allocation in the JVM

You are being given a program called **HeapArrays.java**. This program allocates a number of large integer arrays on the heap when it runs. It is intended to show the maximum amount of memory that can be allocated on the ***heap*** by a Java program before it generates an **OutOfMemoryError** exception.

Compile and run the program. It will stop at intervals after it has allocated a new integer array and show you the heap memory statistics (used, available, maximum that can be allocated by the JVM etc.).

Your tasks for this problem are the following:

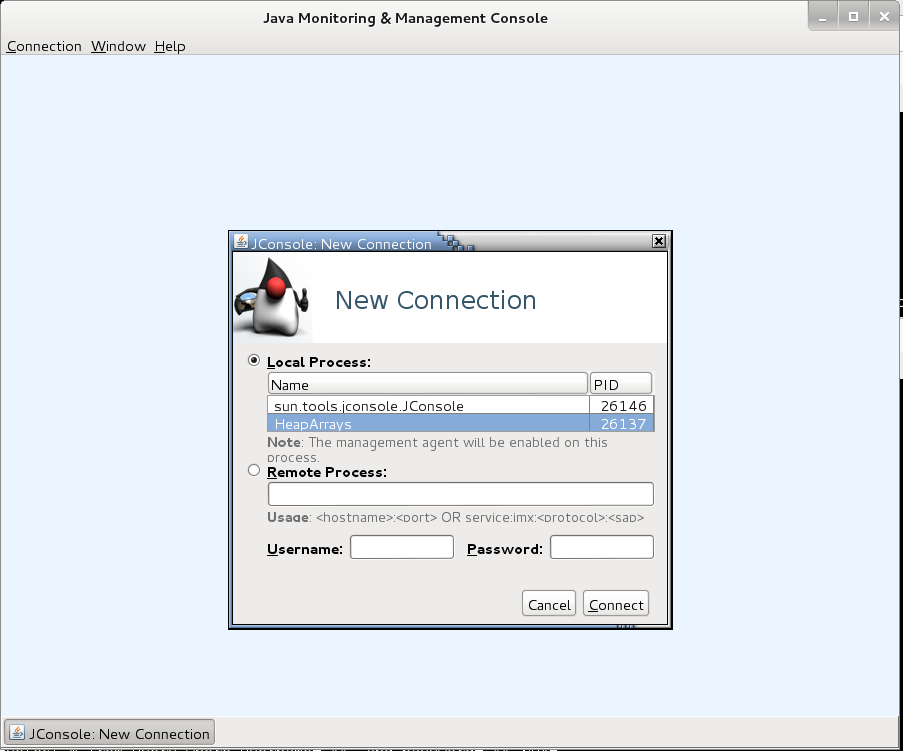
1. Read about how the JVM allocates and manages heap memory. Google for an explanation or see the following link: <http://www.journaldev.com/2856/java-jvm-memory-model-and-garbage-collection-monitoring-tuning>
2. Monitor the memory usage of the program’s heap at run-time using the **jconsole** program (this comes with the JDK)
3. Examine how changes to the amount of memory available to the JVM affect the heap size, and how it is allocated

**jconsole**

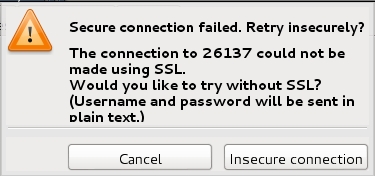
The **jconsole** program comes standard with the JDK. It is a useful debugging tool as it allows the programmer to watch how memory is allocated and threads are created and destroyed as a program is actually running.

To use jconsole do the following:

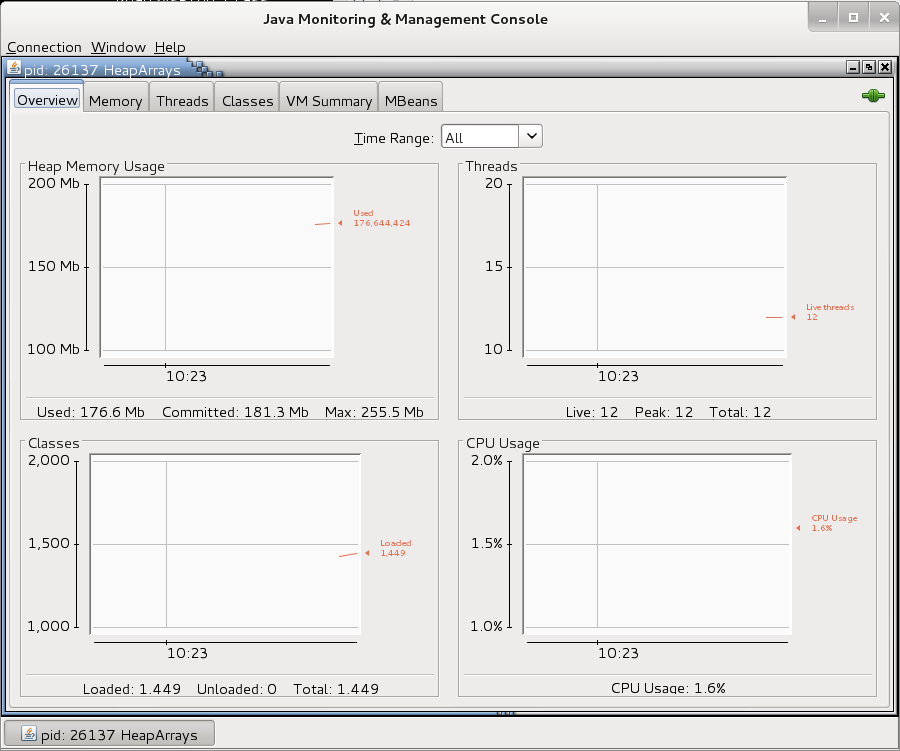
* Compile the **HeapArrays.java** program
* Run the program in the usual way: **java HeapArrays**
* In a separate terminal window type in **jconsole** and hit return
* The **jconsole** window will appear on-screen:



* In the “**Local Process**” list select the ***HeapArrays*** program and click on the **Connect** button
* Click the “**Insecure connection**” button in the dialog that appears:

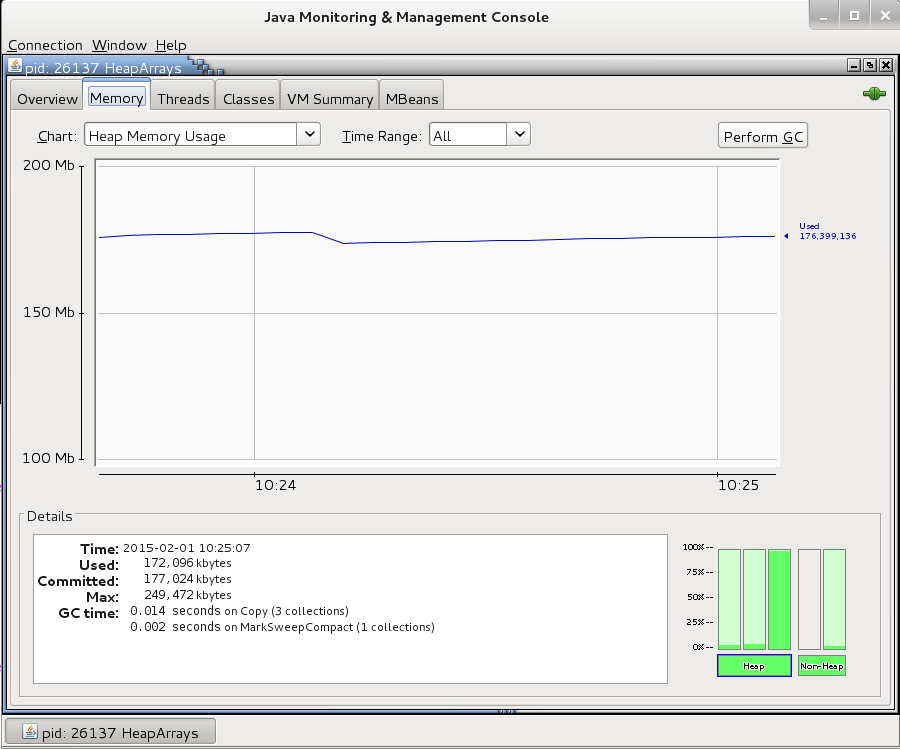


* You should then see the **Overview** page of **jconsole**:



This shows general statistics for how this program is operating and the resources it is consuming.

* Click on the “**Memory**” tab at the top-left:



This page shows the memory statistics for the program, i.e. the memory allocated to the heap and how it is divided up, and the non-heap memory allocation. The three bars over the “**Heap**” box at the bottom-right of the page represent the memory allocation for the “**Eden Space**”, “**Survivor Space**” and “**Tenured Gen(eration)**” respectively. Clicking on one of the boxes shows the statistics for it on the “**Details**” box at the bottom-left of the page.

* Examine the memory allocation for the program as it waits for you to press return having allocated the first really big integer array.
* Press return in the **HeapArrays** program and switch back to **jconsole**. Examine how the memory allocations have changed.
* Continue doing this until **HeapArrays** terminates. Examine the memory allocation at each step.

Run the **HeapArrays** program until you have an understanding of how the JVM is allocating memory with the **HeapArrays** program when it is run using the default heap size (for this JVM it is 512MB by default, although we cannot actually use all of that because of other program overhead requirements).

To cause the JVM to allocate more memory to the heap at run-time use the **–Xmx** command line switch when running the program. This switch allows you to specify different maximum memory sizes for the program other than the default 512MB. Try increasing the heap size for **HeapArrays** and use **jconsole** to examine how that changes the heap memory allocation, and the size of the largest integer array you can now allocate based upon the changes.

e.g. **java -Xmx1024m HeapArrays**

Describe what is happening when you click Perform GC.

# Problem 2: Determine where exceptions are “leaking” information, and fix the problem(s)

Compile the program called **LoginDB.java**. This program checks to see if a user with a particular password exists in a specific MySQL database.

The program takes in five arguments, either at the command line, or on-screen when run if no command line arguments are given:

1. Username of MySQL account to connect as (“root”)
2. Password for MySQL account (“password” – the empty string)
3. Name of the database to connect to in MySQL
4. Username to look for in the database specified at (3)
5. Password for the username specified at (4)

You are being given a MySQL database backup SQL script to use to create the users database containing two users:

* “adam”, password of “adam”
* “bill”, password of “bill”

To create this database using the script in the Kali VM:

* Run a terminal window
* Copy the **users.sql** script into the current directory of the terminal window in the Kali VM (if you haven’t done this already)
* At the command line type in: **mysql -uroot -ppassword < users.sql** and hit return. This will run the script and create the database
* Type in **mysql -uroot -ppassword** and hit return. This runs the **mysql** command line client program. The prompt should change to a “***MariaDB>***”
* Type in “**show databases;**” and hit return. You will see a list of the database in your MySQL instance
* Type “**use users;**” and hit enter. This will switch you into the **users** database
* Type “**show tables;**” and hit enter. You will see a list of the tables in the current database
* Type “**describe users;**” and hit enter. You will see the schema for the **users** table
* Type “**select \* from users;**” and hit enter. You will see the data in the **users** table
* When finished type “**quit**” and press return to leave the **mysql** command line client

Run the program as follows:

**java -classpath .:./mysql-connector-java-5.1.45-bin.jar LoginDB**

The program will prompt you to enter the required data as described at the top of this problem description. To provide this data on the command line you could use:

**java -classpath .:./mysql-connector-java-5.1.45-bin.jar LoginDB users root password adam adam**

The program will output “**Valid**” if there is a user called “**adam**” with a password of “**adam**” in the “**users**” database, and “**Invalid**” if there is not.

***Exception Output***

If the user enters incorrect data, e.g. invalid MySQL username or password, invalid database name, etc. then the program will generate an exception and show the data on-screen.

Your task for this problem is to see what important information you can glean from any exception data generated by the program, and determine how that information could be useful if attacking the system. Try running the **LoginDB** program a number of times, providing different data each time (sometimes valid, sometimes invalid) and see what exceptions are generated and what information they provide.

**Use the information gathered to change the program so that it no longer “leaks” any information useful to an attacker.**