YES, YOU CAN CATCH MEMORY LEAKS DURING TEST

[](https://tier1app.files.wordpress.com/2014/11/waterleak.jpg)Memory leaks are dangerous problems which puts entire application’s availability in to question mark. Most of the time (if not all the time) memory leaks are uncovered in production. This article intends to change this saga. With all confidence I can say that if you follow the below mentioned simple procedure – “Yes, you can catch memory leak during testing phase itself”.

**Step 1: Mimic production volume mix**

Most of the organization do conduct stress tests before releasing the application to production environment. During the stress testing phase, make sure you mimic the load exactly (or close) to production mix. Say suppose if your application gets 50% of volume from API channel, 30% of volume from web channel and 20% of volume from mobile channel then mimic the same volume distribution in your stress testing environment. Also make sure to use same message calls i.e. in API channel if you get 80% invocations for “search” calls, 15% for “booking” calls and 5% for refund calls make sure to use similar mix.

**Step 2: Long running test**

Let the stress test run for a prolonged period (anywhere between 2 – 24 hours). Few times (not always) java.lang.OutOfMemoryError will surface or repeated Garbage collection will be running during this period. These two symptoms will happen only if tipping point is reached. If it happens to be the case, then it’s well and good. You can get applause from your manager. Even if it doesn’t happen, don’t worry next step will get applause from the entire team.

**Step 3: Take Heap Dump of live objects**

Even though in test environment problem may not reach it’s tipping point, but still it would be boiling. If we can capture the boils, it’s good enough to indicate the memory leak exists in the application. This what we are going to do in next two steps.

Once long-running stress test is complete don’t shutdown the JVM. Leave the JVM running for 5 – 10 minutes. Then take heap dump of live objects from the JVM. A heap dump is a snapshot of the memory of a Java process. The snapshot contains information about the Java objects and classes in the **heap** at the moment the snapshot is triggered. Live objects are the objects which have active memory references, which can’t be garbage collected even when GC runs.

There are [few tools to take heap dump](https://blog.tier1app.com/2014/11/16/how-to-capture-heap-dump-jmap/), use one of your favorite tool to capture heap dump. My tool of choice is **jmap**.jmap tool is part of the JDK installation with in the $JAVA\_HOME/bin folder.

Fire the jmap tool, with following options to take heap dump:

jmap -dump:live,file=

where

file-path: is the location where heap dump will be created

process-id: Id of the process for which you want to take heap dump

It’s very important to use “*-dump:live*” parameter. When JMap is invoked with this option then full GC will be triggered on the JVM. After full GC is run, all un-referenced objects will be garbage collected and only live objects will remain in the JVM. After this command is fired, heap dump will be generated in the specified .

**Step 4: Compare with Prior code base heap dump**

Repeat the exact same procedure on the old code base that is handling current production traffic. If you see a considerable increase in the heap dump file size, then it might be because of two reasons:

1. Memory Leak (most likely)

2. New code base might be intentionally holding references to lot more objects (example: Caching)

Irrespective of the case, shoot out a harassing email to development team (just kidding ) with this details and get them to work.

For developers, here is a separate blog on how to handle these sort of harassing OutOfMemoryError problems (i.e. it talks about what tools you should use to analyze heap dumps and how to isolate which part of code is causing the memory leak).