Improving the Security of Garbage Collectors in the Java Virtual Machine

Author: Bc. Patrik Čerbák
Supervisor: Ing. David Kozák, Ph.D.
Year: 2026

Consultants: Francisco Ferrari Bihurriet
Martin Balao Alonso
Thomas Fitzsimmons
and others...

In collaboration with the OpenJDK team at IBM.

Introduction

The aim of this project is to extend the G1 garbage collector in the Java Virtual Machine (JVM for short) to be less vulnerable to memory-related vulnerabilities. These vulnerabilities are still among the most common, and they are often very difficult to debug and identify the cause. This extension should make catching and debugging those vulnerabilities much easier by constantly moving the JVM's memory and using mechanisms for catching usages of the older memory from which it was moved. The idea for this project came from my colleagues at IBM (formerly at Red Hat), Martin Balao Alonso and Francisco Ferrari Bihurriet.

Common Vulnerabilities

When talking about memory vulnerabilities, the most common ones, even today, are undoubtedly *use-after-free* and *buffer overflow*. Both of those are caused by some memory mismanagement happening. A use-after-free vulnerability is created when a pointer points to an object that is later deallocated, but the pointer itself still points to the memory where the object was (this pointer is called a *dangling pointer*). If an attacker gains access to this pointer, they may be able to read potentially sensitive data they would normally not have access to, as the memory might be reused by a different object.

On the other hand, a buffer overflow occurs when a buffer is allocated, and an attacker reads beyond its bounds. They may, for example, get a pointer to the start of an array, iterate through it to reach the end, and continue beyond it, potentially accessing sensitive information stored in objects located after the array.

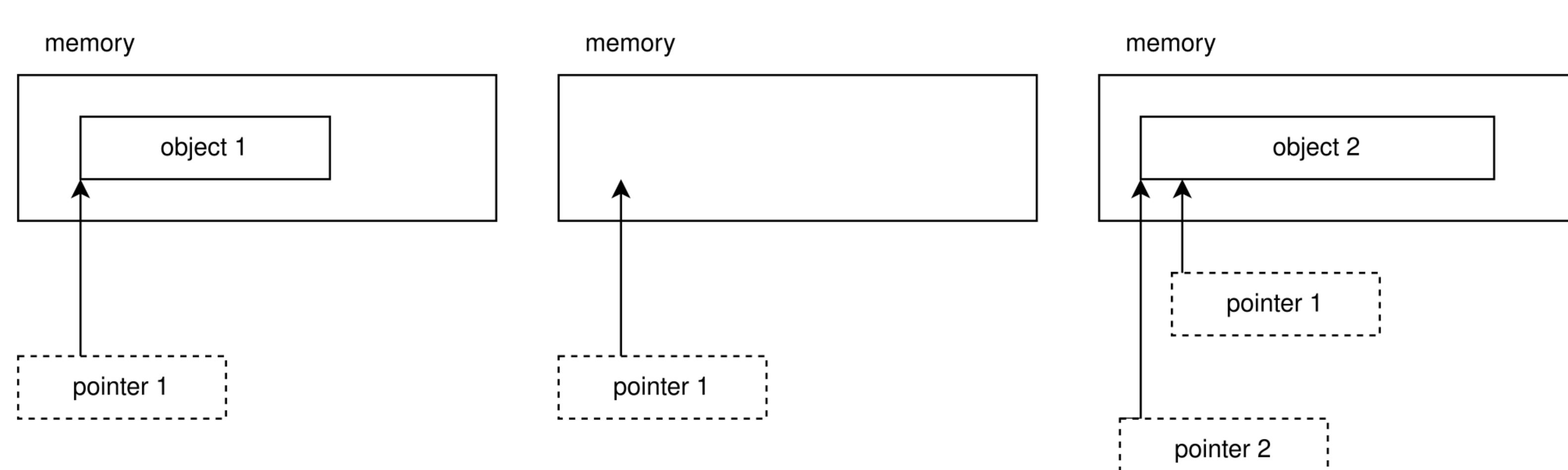

Figure 1: Diagram showing the use-after-free vulnerability.

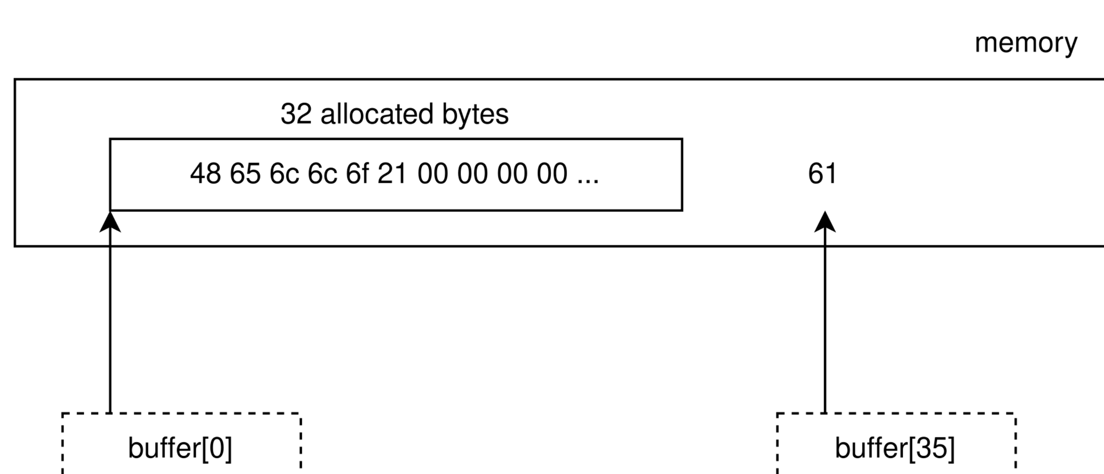

Figure 2: Diagram showing the buffer overflow vulnerability.

The G1 Garbage Collector

The Garbage-First garbage collector (or G1 GC for short) is the de facto default garbage collector in the JVM. It is a very modern garbage collector and ideal for almost all use cases. The important thing about the GC for this project is that it pre-allocates the whole heap (as a one huge chunk) it works with and splits it into multiple structures called *heap regions*.

G1 stores these heap regions in a *free region list* and takes them out of the list when it needs more space for allocating objects. Later, when all objects from a region are deallocated or moved elsewhere (G1 moves objects that survive multiple garbage collections closer together so that it does not have to scan them that often), the region is returned to that list.

Our G1 Extension

What we decided to do, to make the G1 GC more secure against memory vulnerabilities, is to add a mechanism that allocates new memory each time a region is about to be taken from the free region list and moves it there, while removing read and write permissions from the old memory where the region was located (effectively "locking the old memory").

This way, if there is a dangling pointer to the region (or if there is a pointer that overflowed a buffer and is now going through the heap) and it is dereferenced after the memory is "locked", the JVM will catch it. It would also print a stack trace of where the illegal dereference occurred, making debugging this vulnerability much easier.

We call this extension **SanitizeGC**, taking inspiration from Google's AddressSanitizer, which also inspired this project as a whole.

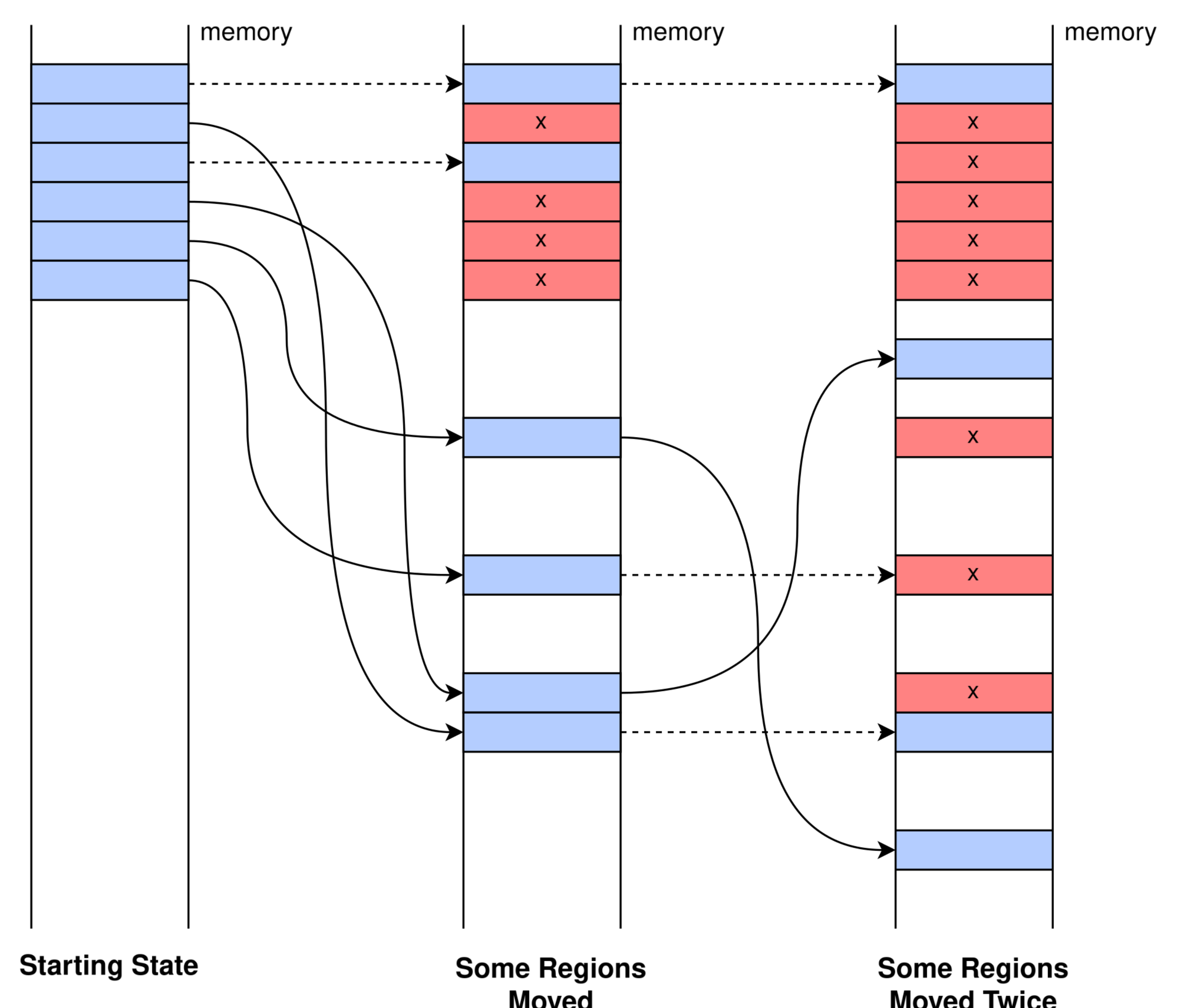

Figure 3: Diagram showing the movement of the heap regions.

Excel@FIT 2026
BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY
IBM