DOI:10.1145/3422590

To view the accompanying paper, visit doi.acm.org/10.1145/3422598

Technical Perspective BLeak: Semantics-Aware Leak Detection in the Web

By Harry Xu

WEB APPLICATIONS ARE at least as likely to leak memory as regular applications. Web leaks can significantly increase a browser's memory footprint, reducing application responsiveness and even crashing browser tabs. Such leaks exist everywhere, on websites that people use on a daily basis—Google Maps, Firefox, Google Analytics, or Airbnb, just to name a few. They are notoriously difficult to diagnose: developers see the growth of memory usage, but where exactly are the statements that cause the growth?

Despite a rich literature of leak detection for regular (Java, C++, Python, and so on) applications, prior techniques do not work well for Web applications where leaks exhibit very different characteristics. For example, the developer may forget to remove certain event listeners and hence these listener objects are still reachable in the heap. While they are no longer used by the application, they still respond to events (for example, when the user uses the mouse on the editor), keeping their states "fresh." As a result, existing techniques that identify suspicious objects based on their staleness (that is, time since their last access)—which have worked effectively on a wide range of traditional applications—would miss these leaks in Web applications entirely.

A key research question here is: What is the right leak oracle that can precisely capture the behavior of leaks in Web applications? In other words, what kinds of objects should be considered suspicious? Once this question is answered, developing a dynamic analysis that finds such objects would be just a step away.

The following paper provides a simple and yet unexpected answer to this question: What distinguishes leaking objects from normally behaved objects is whether their behavior obeys certain high-level semantic rules as opposed to low-level semantics-agnostic access patterns. One clear semantic rule in Web applications is that if a user navigates to a Web page and later returns to the original page, the application's memory consumption should remain (approximately) the same. In other words, the memory consumption for such navigation "round trips" can be used as a leak oracle—if the application consumes significantly more memory when coming back to the original page, the application has a high chance of leaking memory.

Based on this observation, the authors created BLeak, a Web debugger that can help developers quickly find causes of leaks. BLeak uses a user-defined script to drive an application into a loop of navigation round trips. Next, it identifies heap paths that are growing each round trip by differencing heap snapshots. BLeak ranks these paths to find "leak roots," captures call stacks associated with top-ranked leak roots and reports them together to the user for leak diagnosis. With BLeak, the authors were able to precisely and quickly identify important leaks in widely used Web applications including Airbnb and Firefox debugger.

With BLeak, the authors were able to precisely and quickly identify important leaks in widely used Web applications including Airbnb and Firefox debugger.

These results are both impressive and aspiring, particularly in the context of at least 20 years of memory leak research. Prior work uncovers a range of low-level "symptoms" that characterize leaks for a variety of applications. These symptoms are defined at the level of object read and write and often far away from actual causes of leaks. When new applications emerge, these old symptoms no longer correlate with leaks. BLeak takes a step further by exploring semantics-aware diagnosis and demonstrates that simple semantic information provided by developers (for example, round trips) can enable heap tracking that is orders of magnitude more precise than semantics-agnostic symptoms used by conventional approaches.

Looking forward, semantics-aware bug diagnosis and optimization is an exciting research direction, especially given that modern applications and workloads are becoming increasingly complex and diverse. Semantics-agnostic approaches would be either unscalable to large code bases/heaps or unable to adapt to the high diversity in modern workloads. Future work, potentially inspired by the observation made in this paper, will determine how program semantics can be employed to optimize applications in different domains. C

Harry Xu is an associate professor in the computer science department at the University of California Los Angeles, CA, USA.