CSE 503: Assignment 1

Sam Kaufman – kaufmans - 3 April 2019

1. When searching for an open source package available from a repository like NPM or PyPI, etc., it can be difficult to determine whether or not that package contains major security defects or a Trojan horse, or if it will contain either of those in future versions as community contributions are accepted or maintainers change.

The impact of this problem is large. One doesn't have to look far to kind examples of npm packages secretly mining Bitcoin for their authors (or brand new authors), or for widely deployed authentication middleware with comically insecure password handling. And as far as I'm aware, there are no existing tools or techniques to solve this problem, beyond after-the-fact security notifications as provided package managers (e.g. `npm audit`).

I expect this is unsolved because the primary signal used to choose packages, besides a cursory review of the documentation, is the reputation of the author. In my experience, packages published by Google, in part for this reason, will frequently win out over near-identical packages from others authors. And so many of the apparent solutions involve improving information about author responsibility, trustworthiness, and other difficult-to-quantify properties. These problems aren't the sort that many computer scientists feel comfortable with or motivated to attack. Further, it's not entirely clear that other authors are especially well-equipped to judge these things!

Some possible solutions might include independent verification of open source packages by security firms not unlike how Linux distributions audit, to some extent, their packages for long-term support cycles, some amount of automatic formal verification done on every build, or even legal liability for developers who commit this sort of "fraud" or "malpractice" when entering a package into one of these repositories.

2. Some programs run for a long time before control switches to a block which hadn't been previously reached during that execution. Diagnosing bugs in that or a subsequent block can be expensive because the program's runtime extends the user's edit-compile-run loop; this is especially problematic in situations where it is not easy to shorten the program runtime by, for instance, shrinking an input. As an example: perhaps the bug only occurs while the program is running in a datacenter with high-latency resource scheduling, and integration with that resource is critical to the test.

I believe the standard existing approach to this problem is to create a test suite which replaces the problematic component with a stub, but this is difficult in situations where the API-to-be-stubbed is large or the component has complex, non-deterministic output. For instance, TensorFlow frequently takes several minutes to initialize on some hardware platforms, and its API semantics are not completely identical across platforms.

These bugs can be relatively time-consuming and therefore expensive to fix. Further, at least in medium term, I imagine these bugs will become more prevalent as more software moves to datacenters and their associated platform services and heterogenous hardware platforms. This may

also be why this problem hasn't yet been solved: because it is relatively rare when running on traditional architectures in typical environments (i.e. one or a small number of X86/ARM machines) where resource allocation is not especially time-consuming.

A possible solution might be to speculatively run code in a block *after* a long-running loop with partial results whenever possible and issuing a warning whenever that execution would throw an exception or otherwise fail. This might generate a large number of false positives, but could serve as a kind of early warning system.

Time to complete: ~2 hours