The Computing Community Consortium's Response to [White House National Cyber Director Requests Feedback on Open-source Software Security]

October 9, 2023

Written by: Kevin Butler (University of Florida), David Danks (University of California, San Diego), Daniel Lopresti (Lehigh University) and Madeline Hunter (Computing Community Consortium).

This response is from the Computing Research Association (CRA)'s Computing Community Consortium (CCC). CRA is an association of nearly 250 North American computing research organizations, both academic and industrial, and partners from the professional societies. The mission of the CCC is to bring together the computing research community to enable the pursuit of innovative, high-impact computing research that aligns with pressing national and global challenges.

The Request for Information identified many key areas, including core issues and aspects of open source software security. As a group, we first want to commend the effort that went into identification of this collection of sub-topics; each is incredibly important. We also note that a number of the challenges identified in the document are policy challenges. The Computing Community Consortium (CCC) does not play a policy or advocacy role, and so we are unable to comment on questions such as the best use of resources, estimated budgets, or regulatory authority. That being said, we would like to highlight key focus areas and challenges that are either missing or underdeveloped in the document's list.

The Risks of Using Machine Learning to Overcome Security Challenges

Dual use AI is becoming a common concern among computer scientists and the government. In the past, a focus on speed of development and deployment meant that researchers and engineers did not always consider ways in which technology could be exploited and used in unintended ways, resulting in negative impacts on society. Using machine learning to identify exploitable bugs in software - ostensibly to fix them - is no exception: whenever we bring a new capability into the mix, we have to think about how to ensure it cannot be used in unintended ways. In particular, ML for these purposes, including ML that can write code and autonomously identify bugs, could readily be used in malicious ways to develop new attacks, including zero-day attacks. Additionally, ML models might inadvertently or intentionally place backdoors in

code that they are ostensibly securing (e.g., in a manner similar to Ken Thompson's "Reflections on Trusting Trust").[1]

Actors with malicious intentions are going to abuse these technologies no matter what precautions we take. For this reason, it seems more productive to focus on the unintended consequences when well-meaning attempts to use machine learning to secure open source code backfire. These models will acquire powerful knowledge over time and they will become experts at finding dangerous flaws in code. That knowledge could then be extracted for nefarious purposes by adversaries who get their hands on the models. As a result, the security of the models being used to secure open source needs to be assured. Developers must think about how to build AI in a truly trustworthy way that safeguards against someone reverse-engineering the ML to find ways to breach software security. Concerns regarding dual use AI are being discussed in the community and should be amplified in the open-source software security space as well.

Composition

One characteristic of open source software is that developers will often take multiple open source pieces and combine them to create something else entirely. This process of composition, even if individually the pieces are secure, can create new vulnerabilities. As Jaeger et al. write, “In general, the composition of policies that are proven secure may not result in a secure system.”[2] Oftentimes the community is so focused on making the individual pieces secure, we do not think about how they will be combined. The community needs a mechanism to figure out who or how these vulnerabilities will play out when individual pieces are combined.

Incentives

Traditionally, there was very little incentive to work on or maintain open source software. Recently universities have started to recognize faculty work in open source as a form of scholarship counting towards tenure and promotion, taking into consideration the number of downloads of an open source package alongside the citation count of papers the faculty member has published. While this is a great step in the right direction, there are still few if any incentives aimed at maintaining or securing existing open source software.

Many research products are driven by fixed term NSF grants that do not extend to cover continued work and maintenance of the software after it is released. As vulnerabilities come up, as they often do, there is no incentive or available resources to fix it and developers often rely on the community to do it for them.This problem is amplified by the fact that many open source projects have a relatively small number of contributors who truly understand the code and are capable of fixing vulnerabilities. There needs to be financial support and an incentive structure to ensure that open source projects are maintained.

In summary, we have begun to overcome the hurdle of getting professional credit for producing open source. Now the question is focusing that credit on producing high quality (secure) open source and providing appropriate incentives for those who take existing code and work hard to improve it by making it more secure.

---

[1] https://www.cs.cmu.edu/~rdriley/487/papers/Thompson_1984_ReflectionsonTrustingTrust.pdf
[2] https://www.usenix.org/legacy/events/sec03/tech/full_papers/jaeger/jaeger.pdf

Reproducibility and Replicability

A key component of open source is reproducing and replicating the software to build on it to produce new software. While the original product might be secure, the software verification and implementation must be correctly replicated or it could lead to vulnerabilities or adverse consequences in the new product. For example if a piece of secure open source software is released and is not properly replicated, any small change made could create a large vulnerability in the new software. Similarly, with verification if a piece of open source software is released and promised to be secure but someone is unable to replicate the verification - there is no way to double check that what you have built is secure and if you did it correctly. This is an especially vital issue due to the amount of large companies that built their websites based on open source.

Open source, particularly from research artifacts, can sometimes lack step by step instructions to ensure successful installation. For example, a 2016 study of open-source application analysis tools for the Android operating system found that packages often lacked clear documentation, had significant library dependencies making installing them in new environments challenging, and provided no means of verifying the correct configuration of the software.[3] There need to be support mechanisms to ensure safe and accurate replication of open source software and confirm that it functions as intended. The current paradigm for sharing and supporting open source software generally ignores the security implications of the code.

[3] https://dl.acm.org/doi/10.1145/2996358