Position Statement on
*Area 4: Initial minimum requirements for testing software source code*
Requirements for **Static Analysis**

Author: Dr. Stephen Magill (smagill@sonatype.com)
Title: VP of Product Innovation, Sonatype Inc.
Interested in speaking: Yes

# Background

Software velocity has increased steadily in recent years with the adoption of Agile development processes and the advent of DevOps, which automates the software testing and release process. In cloud environments, this velocity can be extremely high. The Accelerate State of DevOps Report shows that at least 7% of companies surveyed release multiple times per day, and companies like Google, Amazon, and Netflix release thousands of times per day (aggregated across their many highly-coupled services). Since the Executive Order on Improving the Nation's Cybersecurity encourages the acceleration of the federal government's adoption of cloud computing services, it is reasonable to believe that such deployment practices will increasingly be adopted in the federal space. This type of on-demand software release requires a new approach to testing. Static analysis tools that run overnight or report their findings to security groups (that then must coordinate manually with developers) operate too slowly to keep up with the pace of software development and thus the potential security risks embedded in that software.

In this position paper, we advocate for *Continuous Assurance*, an approach to incorporating static analysis tools that is directly integrated into the developer workflow and better adapted to Agile, DevOps, and cloud native development processes. This is not a new approach. It has been deployed successfully at Google, Facebook, and other large tech companies. It is also not tied to particular vendor solutions. Anyone can implement this approach in their Continuous Integration process using open source software. And many comercial static analysis tools now support this sort of developer-first workflow integration.

We also advocate for a broader view of software errors. A focus on traditional security patterns such as the OWASP Top 10, while a valuable starting point, risks missing issues that can cause outages and integrity errors that can impact critical infrastructure and lead directly to outages of mission critical applications. Our focus should rather be on software behavior -- does the software do what is intended? Any deviation from intended behavior can cause devastating impacts and any tool or technique for checking aspects of correctness (or finding likely cases of incorrect behavior) is valuable.

# Continuous Assurance

We define *continuous assurance* as a process that incorporates the following key elements:

**Direct integration into the developer workflow.** This could be via the IDE, the build process, git pre-commit hooks, code review, or CI. Since developers are the people that wrote the code

and ultimately are the people that must fix any issues identified in the code, issues that are identified need to be delivered directly to developers whenever possible. Workflows that involve humans triaging results or security teams referring issues back to development teams are too slow to work with high-velocity DevOps processes. With these human-driven reporting-based workflows, software could be operating in the field for hours or days before critical issues are communicated to the development team. And fixing these issues takes additional time. Note that this same requirement of reporting issues directly to developers also applies to infrastructure errors since developers are now writing code (terraform and cloud formation scripts) to automatically configure and provision cloud infrastructure.

**A data-driven approach to monitoring impact.** Data on bugs reported, bugs fixed, and the ratio of the two, termed *fix rate*, must be tracked to monitor and improve the usage of tools. Without visibility into the efficacy of tooling, security and quality assurance teams are in the dark and cannot improve the quality of software being delivered. Fix rates in particular are a key metric that should be measured and be subject to explicit goals. Both Google and Facebook report that they track fix rate as **the** key metric in their software analysis activities. At these organizations, checkers with low fix rates are examined and tuned to provide more accurate or more actionable information to developers. High fix rates are required to support the direct integration of tools into development described previously. Without high fix rates, developers will see mostly useless results, will lose trust in the tooling, and will start ignoring or circumventing the code analysis process.

**A broad focus across issue types.** We term this approach *Continuous Assurance*, not *Continuous Security* or *Continuous Reliability*. All types of errors from security to performance, reliability, and even style are valuable in the delivery of trustworthy software. And approaches like formal verification are even being deployed at companies such as Amazon to ensure that software does exactly what is intended. As an example, a null pointer exception in Java, which does not affect security the way that a memory error in C or C++ does, can nonetheless lead to a denial of service (DoS) attack if the error can be triggered by an attacker. A performance issue, such as using an inefficient collection type can similarly lead to a DoS attack if an attacker can cause elements to be inserted into the collection. More generally, incorrect code can lead to data exposure or data corruption, a common factor in prominent cyber attacks. A general focus on *assurance*, broadly defined, is thus recommended.

## Conclusion

The Executive Order paints a vision for the nation that involves taking advantage of the incredible agility and value that modern software development and cloud deployment processes enable. Adoption of *continuous assurance* processes enables us to embrace this vision while also having systems we can always trust. Large tech companies such as Google, Facebook, and Amazon have adopted these principles for years and they are steadily making their way to the rest of the industry. NIST can accelerate this adoption and protect government systems by ensuring that the government and its suppliers adopt these principles as part of their software development practices.