# Secure Software Development Artifacts

Eric Schulte *Director of Automated Software Engineering* eschulte@grammatech.com

Software supply chains span diverse closed- and open-source production environments. These chains transmit implants and vulnerabilities into software systems. Pursuant to executive order sections 4(e)(ix and x) [1] we propose a range of regulations and requirements to be imposed on secure software development environments. Critical software exists on a spectrum in which the most critical and depended-upon software projects are "too big to fail" and — as with utilities — should be subject to commensurately stringent regulation and requirements (see Figure 1).

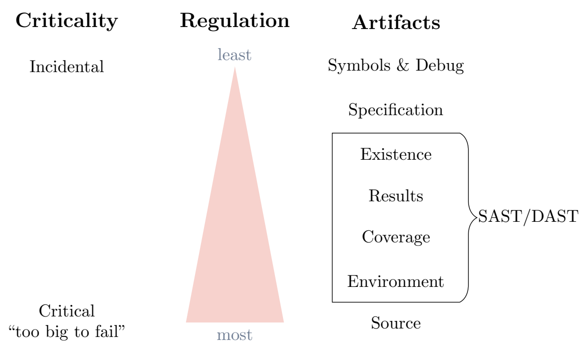

Figure 1: Spectrum of criticality and regulation.

The following artifacts are routinely produced during secure software development: *symbols* and debug information; *source*; *specification*; and Static Analysis Software Testing (SAST) and Dynamic Analysis Software Testing (DAST) *results*, *coverage*, and *environment*. We recommend distributing these artifacts alongside software to secure the software supply chain. We describe the security impact of this distribution and when it should be required along the spectrum of software criticality. These deliveries should be: *Cryptographically signed* by the software developer, *Machine readable*, and distributed in a *Standard format*. We justify these recommendations below.

## 1 Required Artifacts

These artifacts provide the consumer with information about the software and inform the consumers' trust in the software. Increasingly complete artifacts enable reproduction of software evaluations up to enabling consumer-side reproduction of all software security evaluations and safe modification of the software to individualized specifications. This enables a *zero trust* approach to supply chain management. We believe this extreme is necessary for truly critical software.

### 1.1 Software Information

*Symbols, relocations, and debug information* are often removed from libraries and executables by compilers. However, without this information the consumers' ability to inspect, validate, instrument, harden, and modify the delivered software is degraded. Stripping this information is effectively binary obfuscation and may be justified to protect intellectual property (IP) but is anathema to a secure supply chain.

*Source* plays the same role, greatly increasing the ability of the end user to inspect, validate, instrument, harden, and modify delivered software. However the risk of IP theft is commensurately increased. For truly systemically important software we believe open-source development should be required – potentially with compensation to the software owner.

*Specification* documents software's intended functionality and may be written in natural language, dynamic tests, or formal machine-checkable logic. Specifications enable consumers to validate that software implements fully and *only* the specified functionality.

### 1.2 SAST / DAST

For both static and dynamic security testing we propose the following classes of artifacts be distributed with the delivered software. These allow the consumer to enforce their own standards of security and are presented in order of increasing information and control for the consumer and decreasing trust of the producer

*Results* are simple statements that security testing has taken place and may mention a specific standard.

*Coverage* provides information on the degree of the security testing. The most widely used example is test coverage, i.e. the fraction of the software executed by the test suite. It is common to calculate test coverage automatically and regularly check it alongside standard test results. Branch coverage and mutation coverage [2] are more sophisticated forms of test coverage. Mutation coverage may be used to evaluate SAST as well as DAST tools. Specialized forms of mutation coverage such as bug injection provide more probative evaluations of SAST and DAST tools [3]–[6]. This evaluation is particularly critical for SAST tools which are difficult to configure and use [6], and their failure modes are easily misinterpreted as a lack of bugs in the software.

*Environment* (e.g. a test platform, cases and expected results) enables full SAST/DAST reproduction so consumers no longer need to trust the software developer, development environment, or the delivery mechanism. This gold standard should be required for all critical software and is only achievable with open-source software and fully reproducible security artifacts.

### 1.3 Enabled workflows

These artifacts enable the following workflows:

**Validate.** *Specification and SAST/DAST artifacts* allow users to check (with *results* or *coverage*) or confirm (with an *environment*) software conforms to security standards even enabling formal proofs.

**Instrument.** *Source* or *symbols* enable instrumentation to collect test coverage, guide greybox fuzz testing [7], and apply sanitizers to detect common classes of bugs and vulnerabilities [8].

**Harden.** *Source* or *symbols* allow end users to harden their software to protect against many classes of vulnerabilities [9], [10].

**Debloat.** *Specifications and SAST/DAST artifacts* allow consumers to reduce software to the minimum required to meet their needs [11]–[14]. Debloating can reduce complexity and attack surface, and can remove otherwise hard to detect or defeat malicious upstream implants [15]. These techniques are increasingly powerful with *symbols*, *source*, or SAST/DAST environments.

## 2 Distribution

These artifacts should always be accompanied by:

*Cryptographic signatures* serve two important roles. First, they serve as an integrity check for any delivered software or security artifact. Second, they associate that artifact with a developer's identity. Trust is rooted in developer identity.

*Machine readability* and *standard formats*, e.g. the Static Analysis Results Interchange Format (SARIF), are essential to allow automatic processing of artifacts, enabling a consumer's automated software assurance and evaluation to integrate supply chain information.

## 3 Conclusion

Distribution of secure software development artifacts enables consumers to manage supply chain risk. Systemically important software requires full disclosure.

## References

[1] J. R. B. Jr., "Executive order on improving the nation's cybersecurity."
[2] Y. Jia *et al.*, "An analysis and survey of the development of mutation testing"
[3] B. Dolan-Gavitt *et al.*, "LAVA: Large-scale automated vulnerability addition"
[4] P. Hulin *et al.*, "AutoCTF: Creating diverse pwnables via automated bug injection"
[5] S. Roy, *et al.*, "Bug synthesis: Challenging bug-finding tools with deep faults"
[6] V. Kashyap *et al.*, "Automated customized bug-benchmark generation"
[7] A. Fioraldi, *et al.*, "AFL++: Combining incremental steps of fuzzing research"
[8] D. Song *et al.*, "SoK: Sanitizing for security"
[9] L. Szekeres, *et al.*, "SoK: Eternal war in memory"
[10] P. Larsen, *et al.*, "SoK: Automated software diversity"
[11] C. Qian, *et al.*, "RAZOR: A framework for post-deployment software debloating"
[12] M. Ghaffarinia, *et al.*, "Binary control-flow trimming"
[13] K. Heo, *et al.*, "Effective program debloating via reinforcement learning"
[14] C. Soto-Valero, *et al.*, "Trace-based debloat for java bytecode"
[15] K. Thompson, "Reflections on trusting trust"