1. Criteria for designating "critical software”

Submitted by:
Jeff Williams, Veteran Application Security Industry Leader
jeff.williams@contrastsecurity.com
https://www.linkedin.com/in/planetlevel/
410-707-1487

We write to encourage NIST to use a broad definition of “critical software.” There are two parts to the definition. “What is critical?” and “What is software?”

We leave it to others to create rules for interpreting the term “critical.” From EO 13636 (“so vital to the United States that the incapacity or destruction of such systems and assets would have a debilitating impact on security, national economic security, national public health or safety, or any combination of those matters.”

In this position paper, we offer suggestions for interpreting the term “software.” Modern software applications and APIs can be composed of many pieces written in different languages, running in different environments, written by many different teams, and connected with many other environments. The first step in the evaluation of “critical software” is determine the “target of evaluation” – the definition of exactly what is in the “software” and what is not.

NIST can encourage more resilient design by emphasizing these principles:

1. **The “Depends On” test**: NIST should include in the “software” any and all software components, modules, services, platforms, APIs, serverless, etc… where the “software” depends on any security relevant properties of a component. These properties might include traditional security concerns like confidentiality, integrity, availability, or accountability. Or the software might depend on other properties, like performance, usability, or scalability. In any case, we recommend a broad definition here to ensure that critical pieces of software are not left out.

2. **Zero trust**: NIST should follow a “zero trust” approach when considering components of software to include. In a world where systems are highly distributed and interconnected, complex protocols and data formats have dramatically weakened the ability to stop attacks at the perimeter. So every component should be considered on its own, without consideration of external security defenses. By following this principle, NIST will help ensure that systems are built to be more resilient.

3. **Include all four categories of software**: NIST should explicitly include all four categories (listed below) as included if they are used in “critical software.” Modern systems involve many kinds of software, and the lines between them can be complicated. Because software in any of these categories can completely undermine the security of the system, we must consider them all critical. In essence, all categories of software have full privilege to access and disclose all data, corrupt data and software, prevent access to critical capabilities, and become a launching point for further attacks. Below we consider four different categories of software:

Security efforts must be applied equally to all the different types of code in the software supply chain listed below. Attackers will simply shift their efforts to where the defenses are weakest.

A. Software you write: This category involves all the source code and configuration "as code" written for a critical system. This category includes code written by external contractors, their partners, and any other sources. There may be many different repositories of custom code, including:
- Web applications, APIs, serverless
- Client code, including IOT, mobile, and single page apps
- Desktop applications
- Batch jobs
- Customizations and extensions to existing software
- Custom security defenses
- Configuration files, particularly those related to security

B. Software you import: Modern software depends on leveraging large numbers of existing libraries, frameworks, and modules that may be open source or otherwise acquired. This software runs with the full privilege of the application using it, so it needs at least the same level of scrutiny. An even higher level of scrutiny is often justified, as the provenance of this software is often indeterminate.
- Open source and third-party libraries, frameworks, and other modules
- All transitive dependencies required by included libraries
- Drivers, plugins, dynamically loading modules, extensions, and other add on software
- Security defenses

C. Software you run: Many software applications and components are acquired from third parties without insight into whether they are secure or not. If these applications and APIs are used for critical purposes or as part of critical applications, they must also be considered critical. We include in this category platform components as well as third party APIs and services, such as SAAS, that provide little insight into their inner workings.
- Applications and APIs developed by third parties without access to source code
- Platform runtimes, such as Java or .NET platform and included libraries
- Application servers, databases, directories, caches, and other repositories
- Cloud and container images
- Third party web APIs, web services, microservices, RPC services
- External services, particularly those related to security

D. Software you build with: Finally, the software we use to build and operate critical applications and APIs is itself critical. As we have seen in recent breaches, a compromise in any of the tools used to build, package, test, and deploy code could enable an attacker to completely take over an application.
- IDEs
- Build servers
- CI/CD tools
- Deployment tools
- Performance tools
- Testing tools and frameworks
- System administration, monitoring, and configuration tools
- Any other tools running on development machines