Securing embedded software in the real world

Adam Boulton
Chief Technology Officer
BlackBerry Technology Solutions

Resolving software vulnerabilities can often only be done by working at a deeply technical level; the devil really is in the details. Therein lies the challenge of developing a security strategy that can be relied upon to create robust software which is impervious to attack. In real-world scenarios this is a difficult task.

In this article, Adam Boulton, Chief Technology Officer of BlackBerry Technology Solutions, shares his views on strategies to detect and address vulnerabilities - and they may surprise you. The approach he recommends starts with an inspection of the software image in its entirety to identify not just vulnerabilities but trends and deficiencies in the software delivery chain. It involves looking at not just the binaries, but also supporting files included in the delivery, as well as a range of the characteristics and attributes of the software components. This holistic view informs the process of vulnerability discovery, allowing the identification and prioritization of the high-risk items.

Traditional Static Application Security Testing based tooling focuses on source code analysis, however, in our experience there are a lot of negatives to this approach. In particular, source code is rarely available within a software supply chain and the source isn't a one to one representation of the built product. The reason for scanning binaries is pretty clear - you really want to know what is going into the real world because that is what attackers will be able to get their hands on. Another major benefit of taking a holistic approach is being able to analyze compiled software; therefore, we can scan an entire product image and take into account how the components interact. This is very difficult to do during the development phases as not all the teams will work together yet it is possible for their software components to interact without prior knowledge. This is also true for finally checking various non-functional requirements, many of which can be impossible to do without checking the compiled software, such as verifying that the compiler defensive technologies are actually present in the build.

Below we have broken down the broad assessment scenarios and mapped out how SAST analysis fits. A high level, contextual approach is presented, rather than the more traditional method of working from the results first.

A larger, more complex component

A good starting point is to identify the use of insecure APIs which are deemed as such due to their vulnerabilities to malicious buffer overflows. This provides insight into some of the behaviors and business logic of the application. In addition, the inspection of file permission settings is another key area that should be investigated in this case. Assessors need to shift their mindset from searching for specific results for specific issues to gathering a wide range of data that shapes the questions that need asking - and answering. Once the areas of investigation have been mapped out, resolution might require a combination of a manual source code review and possibly a traditional penetration test depending on the exact features being delivered. The takeaway - use automation to determine the inventory and security landscape, assess the non-functional requirements but also have access to relevant data to answer queries that require domain specific knowledge.

An entire operating system and limited time

When presented with a situation like this, the approach becomes very similar to what an external adversary would take which is to use a top down approach to identify the inventory (what applications and resources are present) and assess the image structure (how it is laid out and what the software permissions are). This is a great way to get a lay of the land for some components to leverage and target. Once those areas have been identified, move on to more specific implementation details such as the presence of certain APIs and if robust non-functional requirements, such as compiler defensive technologies, are in place. The difference between the security engineering approach and the adversary approach is that the adversary only has to find one specific issue, while security engineers have to use the same information to determine what is required to thwart different attack classes. For this reason, defense really must be holistic and continual.

This is where the unique features of BlackBerry Jarvis really come into play - the strings dashboard, particularly the email and URL strings, can identify social engineering exposures and unintended access to confidential information. It can even reveal clues to how builds are being handled and if adequate “scrubbing” is in place. This is a basic starting place but extracting and aggregating all these results provides highly valuable insight.
The detection of certificates not only opens up potential risks to infrastructure but also indicates issues with either build sanitization or fundamental security architecture flaws. It can reveal issues such as certification expiration dates or insight into general approaches to cryptography. The presence of risky binaries, such as `telnetd` or `ftpd` suggest similar vulnerabilities. If there is no reason for these applications or convenience tools to be stored within the final product, then the build may not have been assessed adequately and there is an extended attack surface for no reason.

Open source software (OSS) analysis won’t necessarily imply security issues when taken in context but can provide insight into the build process which infers something about the overall software quality.

The takeaway is that large systems are too complex for a single person to understand comprehensively. BlackBerry Jarvis takes has a unique approach by aggregating and cross-referencing hundreds of factors to assess large systems and prioritize components for further assessments.

Scanning products

A large downside of manual assessments, especially paid-for external assessments, is that it is a snapshot in time. A solid month could be spent assessing a product and filing issues of the code at that specific moment in time. Those issues could eventually be fixed, only to be reintroduced a week or month later as the code base grows. Or, maybe new developers join the team halfway through the product and haven't any insight into the previous pitfalls and therefore introduce issues once again. This really raises the need for continual inspection within the software pipeline.

One example, of many, in developing operating systems might include omitting to drop privileges when a new component is added. It would be nice to have a security engineer assess every new component that is added, but clearly that isn't feasible or sustainable. The next best option is to educate the development team and trust they do what is required, but again for various reasons this can be very time consuming and doesn't act as a point of verification. BlackBerry Jarvis is so powerful because it allows you to break free from the one-time assessment approach and move into continual assessment mode.

The takeaway is that software continually evolves, therefore so does risk. Continuous assessments are required to monitor this risk and performing a holistic analysis on a system provides insights into many areas which would have otherwise been missed.

We invite you to review the BlackBerry Jarvis product brochure or email our experts at jarvis@blackberry.com