From Black Box to Glass Box: Making Container Security Explainable

Security scans and automated vulnerability detection efforts have become a cornerstone of modern DevSecOps pipelines and, by proximity, container security. By inspecting container images for known vulnerabilities, developers can identify and mitigate risks before deployment. Yet, for many engineers, these scans can feel like *black boxes* if they cannot understand the underlying logic. The result? Alert fatigue, wasted time, and a potential mistrust of security tooling.

As container adoption grows, it is crucial to rethink how we approach container security as they become more common in software development. The power of containers lies in their ability to encapsulate applications and their dependencies, enabling consistent deployment across diverse environments. However, this encapsulation also introduces complexity in understanding the security posture of containerised applications. Therefore, it is essential to enhance the transparency and explainability of container security tools to ensure they effectively support developers in maintaining secure applications. If developers cannot understand why their software system is vulnerable, automation and portability may quickly turn from a strength into a liability. For that reason, to truly secure our software, we must transition from black boxes to *glass boxes*, i.e., from opaque automation to explainable, transparent, and developer-friendly security systems.

Lessons from Vulnerability Prediction Research

During my doctoral research in information retrieval-driven software vulnerability prediction, I explored how machine learning models can predict potential software flaws before they surface. By analysing historical vulnerability data, these models can identify patterns in code elements and dependencies that tend to lead to security issues.

I found that token-based and embedding-based vulnerability prediction models, which rely on features derived from source code tokens and embeddings, respectively, can effectively identify vulnerable components in software systems. However, their usefulness in a productivity-focused environment could increase dramatically if they are paired with explainability. When developers deem information on vulnerabilities very digestible, they are in a much better position to reason about the decision or leverage their expertise and understanding of the context to contest it. This point marks the crux of the transition from the *black box* to the *glass box* metaphor.

The same vulnerability prediction principle is transferable to container security and presents two key insights relevant to this discussion:

1. **Prediction enables prevention.** Instead of reacting to published CVEs, we can identify risky components proactively.
2. **Explainability fosters trust.** This transparency will help teams act with confidence and accountability, potentially reducing friction between security and product delivery.

Containers encapsulate code, dependencies, and configurations, making them rich sources of security signals: a prospect that improves as we broaden the scope to incorporate security in orchestrated environments such as Kubernetes. By indexing and embedding the effective predictors among these components, vulnerability prediction models can learn to detect risky combinations. Furthermore, with well-implemented information retrieval techniques, these capabilities could be extended, for example, to match current Dockerfile patterns with known insecure configurations (e.g., unpinned dependencies, exposed ports) and provide explainable similarity reasoning.

The Problem with Opaque Scans

Typical container scans can generate hundreds of warnings, many of which can be low-impact, transient, or inherited from base images outside the developers' control. While some tools rank risks by severity, they might not explain their rationale comprehensively, leaving developers wondering:

- Why is this vulnerability severe?
- Is it reachable from my code?
- Does this even affect my running container?

Without context and transparency, scans risk losing credibility. Teams may ignore results altogether or waste time fixing irrelevant issues. Over time, this could lead to *security debt*, bloated CI/CD pipelines, and a loss of developer confidence. Therefore, security tools should *clarify* risk, not obscure it. This lack of transparency is precisely what explainable prediction models can address.

An Example: When "High Severity" Isn't Actually High Risk

Imagine a development team deploying a web application built from a popular base image. A container scan flags a **high-severity OpenSSL vulnerability** in the image. The scanner's dashboard flashes red, the CI/CD pipeline fails, and a developer on the team is instructed to *"fix"* it before merging.

But after investigation, the team realises that:

- Their app doesn't use any feature of OpenSSL.

- The vulnerability only affects a deprecated cipher suite disabled by default.
- A fixed version of the base image isn't yet available upstream.

In context, the "high-severity" issue poses **no practical risk** to their running container, yet it **blocks the deployment** and potentially **erodes trust** in the scanning tool.

Why Context Matters

Without context, the scanner's warning appears urgent; however, in reality, it's blind to reachability and dependency scope because it lacks insights into that specific web app's attack surface and cannot assess whether the vulnerable component is actually used. In this scenario, context can transform a binary "high vs. low" severity rating into an **explainable and actionable insight**.

Explainable prediction models could make that distinction explicit, showing *why* something is severe and *how* it affects the container, rather than simply flagging it, which could potentially save time and frustration.

Complementing Prediction with Explainability

Explainable AI (XAI) techniques, such as Local Interpretable Model-Agnostic Explanations (LIME) and SHapley Additive exPlanations (SHAP), make complex models interpretable and have seen applications in domains such as healthcare and finance.

Vulnerability prediction combined with XAI in software security can potentially reveal to developers which components, whether code elements, API calls, or dependency chains, contributed most to a vulnerability flag. Instead of "fix this because the scanner said so," engineers get, "fix this because these insecure patterns have historically led to exploits."*

This pairing would bridge the gap between automation and human reasoning, potentially transforming security tools from *reactive gatekeepers* that sometimes block software deployments, without explanation, pending a security review that may or may not be relevant, into *proactive mentors* that teach as they protect. Proactive mentorship in this context refers to the idea that explainable security tools have the potential to do more than just flag issues; they can offer developers passive learning opportunities through transparent explanations, helping them internalise secure coding practices over time.

What This Would Look Like in Practice

Imagine a hypothetical scenario where a developer uses a container security tool equipped with an *Explainable Scan Mode*.
Instead of a long list of CVEs, they might see contextualised insights like:

- **Root cause:** Deprecated SSL method in `Auth.java`
- **CWE:** CWE-295: Improper Certificate Validation
- **Confidence:** 0.87 (based on similarity to 2,000 known vulnerable methods)
- **Key contributors:** `verifyCertificate()` , hardcoded token, unsanitised input
- **Location:** `/app/src/main/java/com/example/Auth.java:45–78`

This explanation not only shows the *what*, but also reveals the *why*, meaning that the same developer in this scenario would be in a better position to recognise and avoid similar pitfalls in future code. Over time, such transparency fosters both productivity and learning.

A Future of Trustworthy Automation

The future of container security lies in trustworthy automation, i.e., systems that can make intelligent decisions while keeping humans in the loop. Explainable vulnerability prediction models could be the key to achieving this balance.

Imagine a CI/CD pipeline where every flagged issue comes with a narrative: *Here is what triggered the alert, here is how it compares to past incidents, and here is the minimal patch you can apply.*
Developers could spend less time trying to understand complex security-related logs and more time improving code quality. However, to reach that point, we must view security as not just a technical problem, but also a cognitive one. Developers need to *trust* their tools to use them effectively. A transparent security system would empower developers rather than police them. It will invite curiosity, prompting a question like *"Why was this flagged?"* rather than a frustration-induced one, such as *"What does this even mean?"* This shift in mindset will be crucial for fostering passive learning.

Furthermore, explainability will also encourage better cross-functional collaboration. Security teams can gain insight into the process of generating predictions, while developers can have the agency to validate and refine them. Over time, this feedback loop could strengthen the underlying vulnerability prediction models, creating a virtuous cycle of trust and improvement that ultimately benefits the entire software supply chain by enhancing its security posture.

Challenges and Trade-Offs

It is essential to clarify that both predictive models and explainability techniques have limitations. Machine learning models heavily depend on the quality and representativeness of their training data, i.e., the "garbage in, garbage out" principle. So, if the training data contains biases, inaccuracies, or is not representative of real-world scenarios, the model's predictions will likely be flawed. Additionally, predictive accuracy is never perfect; false positives and negatives will inevitably occur. Therefore, these fundamental limitations associated with predictive models remain and must be carefully managed. There is further interesting discussion on the challenges of applying vulnerability prediction models in this study from Microsoft.

On the explainability side, it also comes with trade-offs. Revealing too much model logic could expose proprietary algorithms, confuse developers with excessive detail, and incur additional computational costs. So, the goal should not be *complete transparency* but *actionable transparency*: enough clarity for developers to understand and act without being overwhelmed. On that note, organisations must also manage privacy concerns carefully. Explanations should not expose sensitive source code or dependencies in shared environments. Balancing security, usability, and explainability will be key. Docker's ongoing investment in software supply chain security, for example, provides a strong model for handling such risks responsibly.

Conclusion

Container security tools have evolved rapidly, but their explainability has not necessarily kept pace with this evolution. Opaque vulnerability scans may tick compliance boxes, but they do not necessarily build confidence.

Moving from black box to glass box is about more than visibility; it is about empowerment. Explainable vulnerability prediction models can make security an enabler, not an obstacle. They can turn security from a burden into a dialogue, bringing us closer to a world where container security is not just automated, but understandable, collaborative, and ultimately trustworthy.

By embracing explainability, we can realise a future where security is not just a checkbox but a shared responsibility, understood and embraced by all developers. This shift will not only enhance the security of containerised applications but also foster a culture of continuous learning and improvement among software professionals, ultimately leading to more resilient software systems.