BEST PRACTICES FOR INTRODUCING JFROG XRAY INTO YOUR DEVSECOPS PROCESS

Start Implementing a DevSecOps Process
in Your Organization

Publish: March 2021

TABLE OF CONTENTS

INTRODUCTION

DevSecOps is a relatively "new" culture which involves both Dev and Ops in the process of handling security and compliance issues as part of the Software Development Life Cycle (SDLC). From the first line of code in the Integrated Development Environment (IDE) all the way through to the delivery of the application to production.

Implementing a good DevOps methodology is tough, but getting DevSecOps right is a bigger challenge. DevSecOps has a broad scope and is the philosophy of integrating different security practices within the DevOps process. The goal is to minimize security risks at each stage of the SDLC. We have seen the effects of what a targeted attack can do to large enterprises with the recent SolarWinds breach. Security isn't an isolated matter, and identifying vulnerabilities is inseparable from software development.

The continous growth in use of open source software (OSS) components, exposes code bases to potential hidden vulnerabilities and license compliance violations. The question is... how do you safely use OSS components in a software development pipeline, without putting production code at risk. Monitoring and mitigating risks within OSS components is handled with a DevSecOps tool called Software Composition Analysis (SCA).

JFrog Xray is the Software Composition Analysis (SCA) tool that monitors and provides insights into your (OSS) packages regarding security and compliance. It is an integral part of the JFrog DevOps Platform, and is natively integrated with JFrog Artifactory, which stores and organizes all your software artifacts. By taking the best of both solutions you can achieve security, compliance and traceability for your software artifacts.

This white paper will guide you through the best practices for implementing a DevSecOps process, with JFrog Xray, into your organization. It includes the following three parts:

1. Increasing the traceability of artifacts within JFrog Artifactory by adopting a recommended repository structure and relying on build info.
2. Defining security and license compliance behavior specifications by organizing policies and watches within JFrog Xray.
3. Integrating JFrog Xray into your security and compliance workflow.

JFROG ARTIFACTORY: SOFTWARE ARTIFACT TRACEABILITY

As a universal artifact repository manager, Artifactory allows you to organize your artifacts within repositories in order to:

- Monitor your artifacts during their lifecycle.
- Understand which dependencies versions are consumed during your builds.
- Apply security policies and restrict the access to artifacts based on a specific profile.

To improve your artifact traceability, you'll need a good repository structure and build info.

Repository Structure

There are 2 common patterns for achieving repository organization:

1. Monolith artifact repository: all teams share the same repositories.
   For example:
   - 1 Remote repository for all Java teams.
   - 1 Local repository to store all Docker images.
2. Dedicated artifact repository: each team has their own set of repositories to isolate their own applications and dependencies.

Below are the pros and impacts of these 2 patterns in terms of artifact traceability.

Monolith artifact repository

This pattern is recommended when starting out with Artifactory. Keep in mind that as the number of projects you onboard increase, the more difficult it will be to maintain and implement artifact traceability. Eventually, you'll need to switch over to a dedicated artifact repository.

Benefits	Impacts
Configuration: - Super fast to set up. - 1 remote repository + 1 local repository per package type used by multiple teams.	Traceability and Security: - Cleanup is harder to perform as you don't want to delete used dependencies. - Access restriction has to be implemented on repository paths which can be hard to maintain when dealing with multiple projects.

Dedicated artifact repositories

This pattern is recommended if you plan to use Artifactory as the central repository manager within your organization. It will allow you to scale alongside your organization.

Benefits	Impacts
Traceability › Remote repository per project: you know which dependencies were consumed. Isolation › Easier to secure and grant permission at the repo level (instead of paths within a repo).	Configuration › More repositories to manage

Good to know!

- Artifactory uses checksum based storage (no duplicates) for improved performance and reduced storage.
- Mask the complexity of your repository structure via virtual repositories and permission targets.
- Automation via Rest API helps a lot when managing hundreds of repositories

Build Info

Build-info is all the information collected by the build agent about the build. The build-info includes the list of project modules, artifacts, dependencies, environment and variables. When using one of the JFrog clients to build the code, the client can collect the build-info and publish it to Artifactory. When the build-info is published to Artifactory, all the published details are accessible from within the Artifactory UI.

Benefits	Impacts
Traceability › All binaries generated and consumed are recorded into this Bill Of Materials (JSON file).	Configuration › Modify your CI pipelines.

Good to know!

- Use Artifactory CI plugins and JFrog CLI to easily generate build info..

Best practices with JFrog Artifactory

Development teams are recommended to use dedicated repositories and build info. This guarantees full traceability of artifacts, including:

- Tracking which dependency versions were used to build a specific version of your product.
- Tracking software during its lifecycle/delivery process.
- Isolating artifacts per scope (i.e. projects, teams, artifact maturity level), which is useful during cleanups!

Learn more about Artifactory best practices >

JFROG XRAY: OPEN SOURCE SOFTWARE SECURITY AND COMPLIANCE

Now that you've organized your Artifactory and can fully trace your artifacts, let's see how to implement your OSS security processes via correct organization of your Policies and Watches with Xray.

JFrog Xray relies on the following features to implement your security process:

- Policies
- Watches
- Violations

You can define a set of policies and watches based on the internal legal and compliance policies of your organization and your chosen security process.
Example organization:

- Each dev project is represented by a release manager/tech leader.
- The DevOps team is responsible for the CI/CD tools.
- The Security team is responsible for implementing a security process.
- The DevOps and security teams are both global or have cross projects.
- DevSecOps (DevOps + Security teams) implement the security process.
- Security is responsible to define the policies and DevOps implement them in Xray.

Organizing your policies will depend on your internal security processes. More on this in the violations workflow section below. It is recommended to set a watch per project and per maturity level (cross project).

The following outlines 2 types of policy structures and patterns:

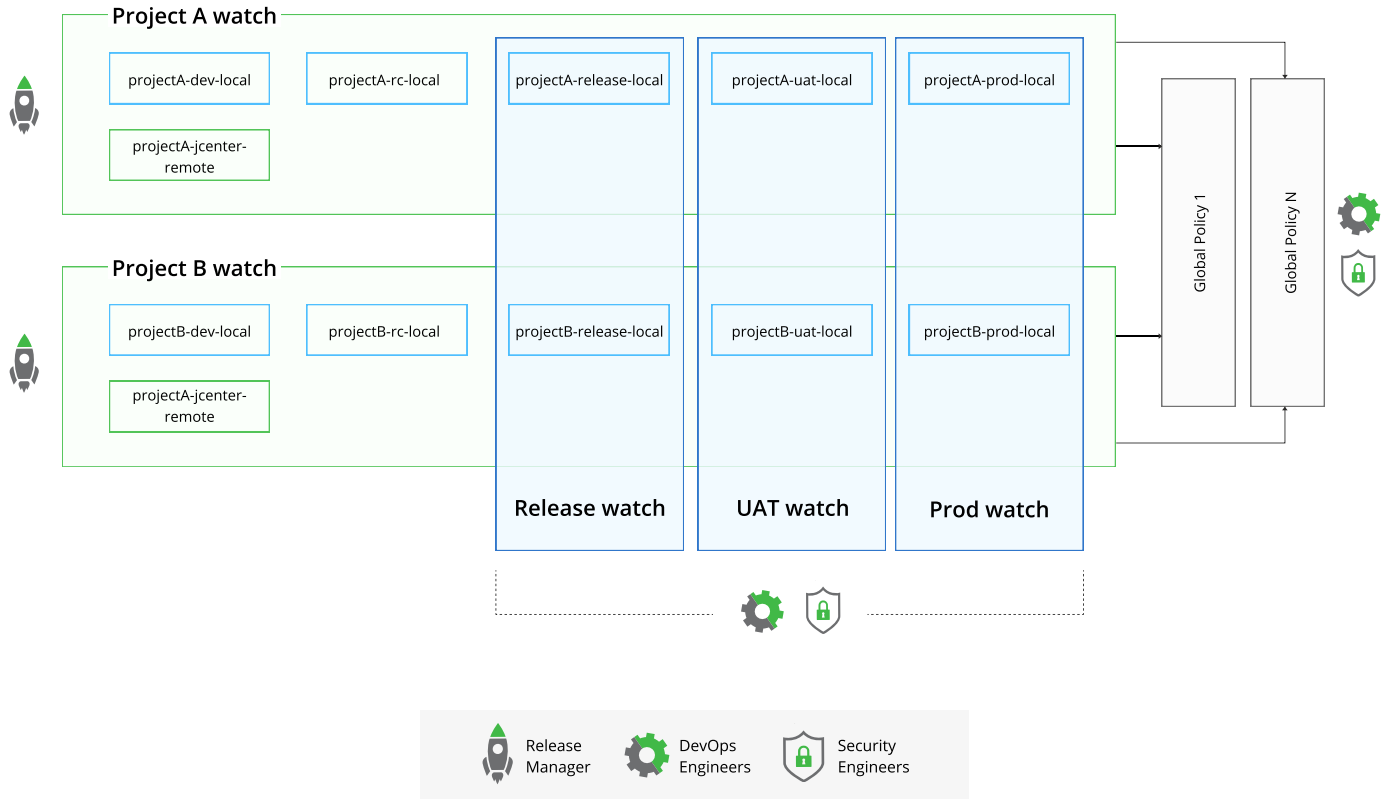
- Global policies, set for the entire organization.
- Dedicated policies, set per project/application/team.

Pattern 1: Enterprise level policies

This pattern describes policies that are global to the entire organization.

As shown in the illustration below, the enterprise level policies pattern implies that projects A and B:

- Share the same definition of high, medium and low severity violations (specific range of CVSS score).
- Use the same post actions, such as block download or fail build.



Recommended practices:

- Each release manager / tech leader should manage the violations for their own project (list and ignore) via watches.

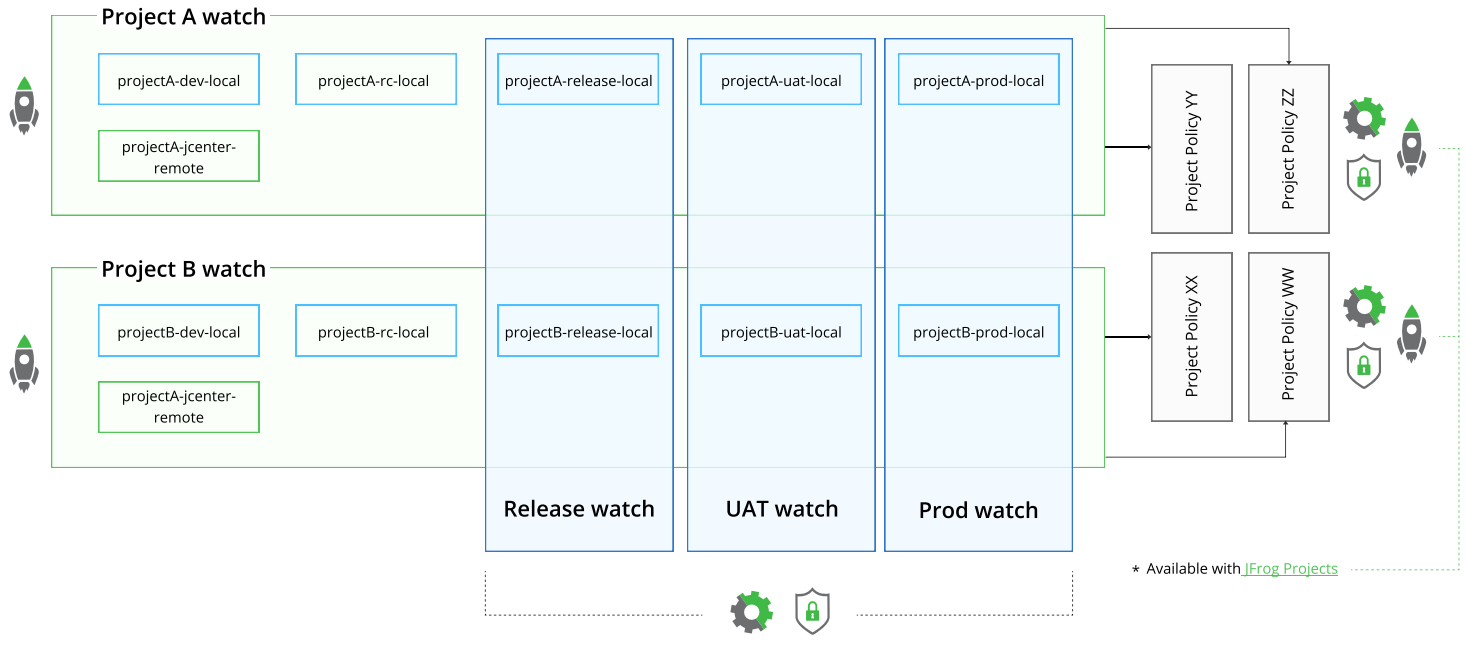
Cross project watches help DevSecOps teams to monitor all projects.

Pattern 2: Project level policies

This pattern describes policies that are dedicated to specific projects.

Projects that are more exposed than internal ones, for example internet front facing applications, require stricter security policies.

As shown in the illustration below, the project level policies pattern implies that the DevOps team builds **seperate policies based on each project / business** unit and makes sure they are compliant with the security team's instructions.



Recommended practices:

- Each release manager / tech leader manages the violations for their own project (list and ignore) via watches.
- Cross project watches help DevSecOps teams to monitor all the projects.
- Dedicated policies per project are created and maintained by DevSecOps teams.

Best Practices with JFrog Xray

Additional guidelines to keep in mind:

- REST API can help you create/update/delete policies and watches. Automate the creation and update and use your own portal to restrict the access to policies.
- Email listing at:
  - The policy level - aims to inform the policies managers (DevOps or security teams).
  - The watch level - aims to inform a group of users at the project level.
- Consider restricting ignore violations to your release manager or tech leader to make sure it will be properly checked and handled.

- Indexing remote repositories will allow Xray to scan 3rd party dependencies as soon as they get requested by the development teams.
- Indexing local repositories will allow Xray to scan the OSS used in your applications during the whole delivery process even after they get deployed to production.
- Indexing the build info will allow Xray to notify and/or fail a build during your CI pipeline in case a violation was identified in a build or any runtime dependencies.

Learn more about Xray best practices >

VIOLATIONS WORKFLOW

Keep in mind that JFrog Xray is a Software Composition Analysis (SCA) tool and NOT an issue tracker. It can definitely be part of your violation management, but will have to be integrated with other tools to entirely cover your security and remediation process.

Introducing a new process is exciting and also challenging as you have to cover all the possible workflow and use cases.

The following is a (non-exhaustive) list of questions your violation workflow should address:

1. When and how you should notify users of a violation?
2. Who should review the violations?
3. Where you should review the violations?
4. How to prioritize security and compliance violations?
5. How to ignore a violation?
6. When / How often should you review violations? and with whom?
7. To block or not to block a download?

See the answers to these in the recommendations section below.

A violation review should be introduced. It could be compared to a “sprint retrospective” in the agile methodology since it requires different parties (i.e scrum master, dev team, product owner), you have to follow rules and take actions at the end of the meeting.

The main barriers between stakeholders (i.e development teams, security, legal teams) include:

- Language/Terminology
- Priority
- Availability of each stakeholder

1. When and how you should notify users of a violation

"Shift left" by scanning dependencies as soon as possible during:

- Development using Xray's IDE plugins
- Local builds by indexing Artifactory remote repositories.
- CI builds by indexing Build Info.

TIP:

Xray implements continuous scanning which means that scans are triggered by events:

- Newly cached binaries into remote repositories
- Newly uploaded binaries to local repositories
- Vulnerability and license database updates. The scan will only affect the binaries impacted by the updates (No full scan)

TIP:

While emails and instant messaging (such as Slack of MS teams) are a great media for communication, they might not be suitable when hundreds of alerts are sent. It is rather advised to rely on the following to warn your developers during the development process:

- The IDE plugin - To identify licenses and vulnerabilities the earliest!
- The CI integration - To get the list of all violations during your CI pipeline and stop it if necessary.

TIP:

Don't slow down the development teams and stop the pipeline before any major steps in your delivery process, for instance before merging a PR to the master/develop branch.

2. Who should review the violations

Developers can quickly detect and analyze vulnerabilities and license violations via the IDE plugin and CI scan, and fix them. For the ones which don't have a fix yet or related to compliance, you may want to let the release manager/tech leader handle them with the security team as those violations will take more time to investigate.

To not slow down the development phase, violations can be ignored for a period of time (snooze) or until the next scan.

TIP:

Implement a clear workflow when:

- No fix is available for a vulnerability
- No license attached to a package

TIP:

If you want to manage violations in Xray, make sure you standardize the comment ⟶
Come up with a pattern. Then retrieve and filter the list of ignored violations using the Rest API.

3. Where you should review the violations

JFrog Xray can list violations at different levels: artifact, build info, watch. But, its role isn't to manage the entire violation workflow which might be more suitable for an issue tracker.
For instance, your violation workflow could lead to creating a ticket in your issue tracker:

- If Xray doesn't provide enough remediation information regarding a security violation or the license couldn't be detected, you could list them in your issue tracker.
  Or
- For any violations raised by Xray.

You may also want to use the JFrog Platform built-in reporting for reviews which are customizable. You can export them in different formats (JSON, PDF, CSV) via the UI and REST API.

TIP:

Use the following REST API endpoints to extract vulnerabilities, licenses, violations:

- Build Summary
- Artifact Summary
- Violation Summary

4. How to prioritize security and compliance violations

Start examining the more critical violations first.
Watch this video about Xray onboarding best practices >

5. How to ignore a violation

JFrog Xray ignore rules allow you to whitelist, ignore or accept security violation rules, in order to filter out unwanted violation noise.

Viewing violations in Xray can be done on 3 different levels: watch, build and artifact.

A watch contains a combined list of violations from multiple resources (including repositories, builds and release bundles). Although having a global view of all violations is convenient, when defining ignore violations, you want to drill down to the specific view depending on what you are trying to do. For example, to ignore violations on a specific build number, you would create an ignore rule at the build level. .

Learn more about how to ignore violations >

6. When / How often to review violations and with whom

The more critical and exposed your projects are, the more often you might want to review violations in your delivery process. Xray provides a real time view of violations and you can easily extract them via REST API.

7. To block or not to block a download

Your security team might certainly want to go with blocking any artifact with security threats. However this can drastically slow down the development process (and potentially annoy your dev teams) as blocking a dependency download could consequently fail a build.

Block download can be enabled on:

- Remote repositories to prevent developers downloading vulnerable or non-compliant dependencies.
- Local repositories to prevent deploying vulnerable applications to a runtime environment.

If you wish to block a download, it is important to make sure to train your teams to detect this behavior!

TIP:

The *block unscanned* artifact option temporarily blocks the artifact until the scan is complete. Make sure to set this properly when enabling it.

Conclusion

This white paper provides guidelines for integrating the JFrog Platform, including Artifactory and Xray, into your organization to implement and leverage your DevSecOps culture.

You can achieve full traceability for your artifacts and build a security process with the JFrog Platform. This can be done by adopting the right repository structure and relying on your build info in Artifactory. You can also organize policies and watches to match your internal security and compliance structure in JFrog Xray. The most difficult part will be to define your entire security violation workflow to achieve DevSecOps within your organization which will impact both your development, operations and security and compliance teams.

Whether you are already working with Artifactory and Xray, or just starting out, give these DevSecOps best practices a try.

START FOR FREE >