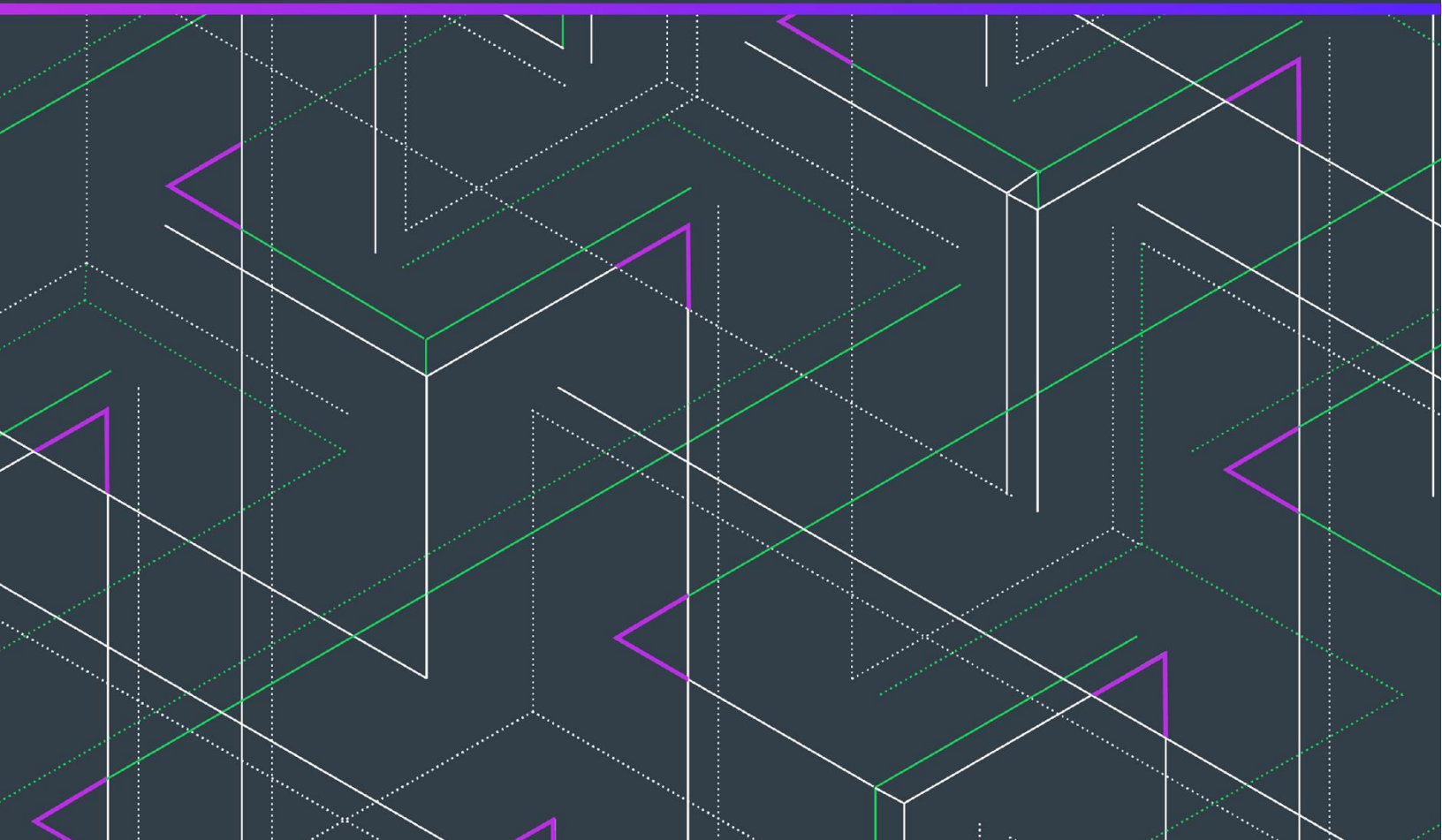
Revenera's Application Security Incident Response Process Overview

Version 1.0

June 17, 2021



Contents

Introduction

This document has been developed to assist our customers, and in turn, our customers' customers in better understanding how we manage externally reported application security issues in our products. This is NOT intended to be a comprehensive document but should act as an overview of our Product Security Incident Response Team (PSIRT) procedures along with an explanation of how we classify and respond to reported application security issues.

Highlights of Revenera's Security Program

Revenera's security program is designed to mitigate application security issues during the development lifecycle and allow us to quickly respond to any reported security issues for released applications.

Internal Cybersecurity Committee

Revenera has an internal cybersecurity committee comprised of a cross-functional group of security-minded individuals to help shape and mature the Company's information security and privacy program. The committee is represented by leaders from Security, Legal, Engineering, and Product Management teams.

The committee is charged with two critical responsibilities:

- to protect our intellectual property and our customer's data
- to provide assurances that we do so effectively

Corporate Security Policy

Revenera has developed a corporate security policy that establishes guidelines regarding security scan tools, scan frequencies, incident response procedures, and suggested remediation timelines.

Security Research and Advisory Support Team

Our security research and advisory support team is comprised of experienced security analysts who are tasked with the following:

- support product teams with assessing security vulnerabilities and their impact to the respective application
- conduct security audits using various security tools to assess the overall security posture of various products
- assist product teams with incident responses as needed when they arise

Security Testing

As part of our security program, several types of security testing techniques are used to provide a comprehensive assessment of all applications during the development process.

Static Application Security Testing (SAST)

Static application security testing (SAST) is a white box method of testing. It examines the code to find software flaws and weaknesses such as SQL injections and cross-site scripting (XSS), along with others listed in the Open Web Application Security Project® (OWASP) top 10. The OWASP Top 10 represents a broad consensus about the most critical security risks to web applications. SAST is typically performed early in a DevSecOps pipeline, typically during the build phase. This allows for the discovery of security issues very early in the development lifecycle when remediation is less complex and least expensive.

Dynamic Application Security Testing (DAST)

Dynamic application security testing (DAST) is a black box testing method that examines an application as it is running to find vulnerabilities that an attacker could exploit. This technique does not require access to the source code of the application, but instead mimics a malicious user to try to exploit the application at runtime. DAST tools tend to live farther to the right of SAST in the DevSecOps pipeline, typically during the test phase, and works with code that is a release candidate or runnable.

Software Composition Analysis (SCA)

Software composition analysis (SCA) is a solution used to document an application's dependency on open source and third-party software which typically comprises 50%-80% of modern software applications. SCA supports both a proactive disclosure use case as well as a scanning use case to validate the software bill of materials (SBOM). It is used to identify security vulnerabilities found in open source and third-party components that are incorporated into applications whose code is maintained outside the organizations. SCA can be performed at various points throughout the DevSecOps pipeline. This is typically done during the code and

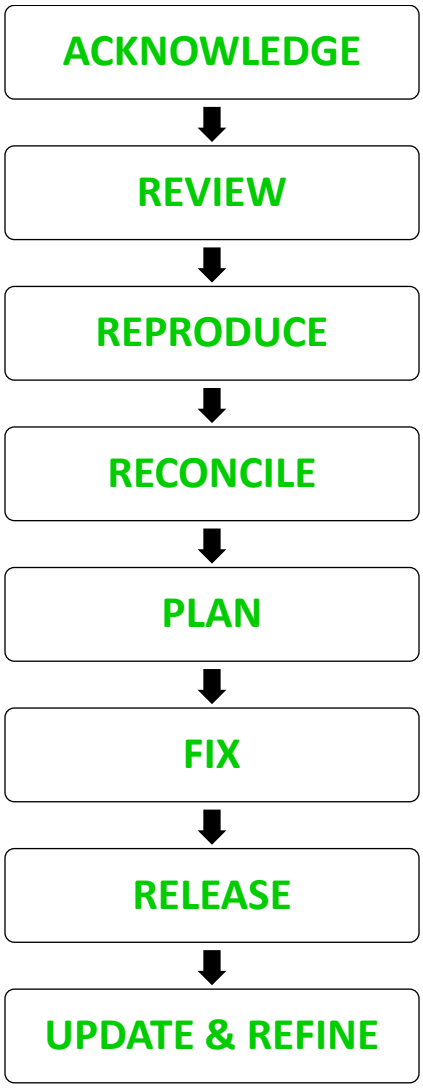
build phases for frequent early indication of issues and during the test phase for a more comprehensive analysis later in the process.

Manual Security Audits

Manual security audits are periodically performed by security experts using various industry tools. Typically, these audits are performed on an annual basis and are more comprehensive than automated test tools and discover a broader set of security defects.

Internal Process for **Reported** Security Issues

The following diagram depicts Revenera's Product Security Incident Response Team (PSIRT) process:



Acknowledge Inbound Security Issue

When we first learn of a reported security issue, it is imperative that we promptly acknowledge the reported issue so that the reporting party knows that we are actively working on it. Issues are typically reported in one of three ways:

- Directly via Flexera's PSIRT page – https://www.flexera.com/about-us/contact-us/report-vulnerability.html
- Directly via the Revenera community site (requires login) – https://community.flexera.com/t5/forums/postpage/board-id/@support
- Directly via a Revenera support call
- Indirectly via a customer success manager (CSM)

Review Issues with Reporting Party and Obtain Necessary Information

After acknowledging the reported security issue, an initial call with the reporting party is crucial to be able to obtain important information to better understand the context of the issue.

To that end, the following questions will allow us to try to reproduce the findings on the appropriate set of files:

What type of issue is being reported?

- Open source vulnerability (CVE) and/or security defect in our proprietary code?

What was scanned to produce the discovered issues?

- Revenera product name and version along with which files were scanned.
- We want to make sure that we are not wasting time assessing issues in the wrong files. Understanding exactly which files were scanned will help us focus on the problem at hand more quickly.

Which scan tools were used?

- Understanding which tools were used (SAST/DAST/SCA) to discover the reported security issue along with the tool's vendor will help us correlate our results and produce a unified response to the reporting party.
- The following information is useful: scan type, vulnerability id or name, vulnerability category, CVSS score, severity, impacted file paths, OSS components and versions, etc.

Reproduce Finding via Internal Scanning

Once we understand which products are impacted and which specific files were scanned by the reporting party, and what type of security issues are being reported, we can correlate the reported issues to our internal scans.

Often, we will already have recent scans available to analyze for the reported issues. If scans are not available either due to the amount of time that has passed since the last scan or due to scan settings or codebase differences, a new scan may have to be performed.

Once the scan results are available, reconciliation can be performed.

Reconcile Reported Results Against Revenera's Internal Findings

Typically, the following outcomes are reached based on reconciliation of reported results by an external party with Revenera's internal findings:

- **Confirmed Finding**: Revenera's internal result matches that reported by an external party
- **False Negative**: additional finding by Revenera not reported by an external party
- **False Positive**: finding reported by an external party which cannot be identified or confirmed by Revenera along with additional findings by Revenera which show up on our scans but are not valid or exploitable in the product itself

Develop and Report the Remediation Plan

Once reconciled results are available, Revenera will again engage with the reporting party to review the findings, explain any discrepancies in the form of false negatives and false positives, as described previously, and discuss the remediation plan to address the confirmed findings.

It is not uncommon to require several working sessions to review the results in detail and arrive at a mutually agreed-upon remediation plan and timeline.

Implementation Work to Address Agreed-Upon Remediation Plan

Once an agreed-upon remediation plan has been defined, the Revenera Engineering team will prioritize the implementation work into its product backlog to address the security issues.

Release Fixed Product

Depending on the amount of work and the existing prioritized backlog, the product may be patched with the fixes, or the fixes may be incorporated into a future complete release of the product. In some cases, it may require several releases to fully address security issues.

Update Security Testing Process as Needed

With each cycle, we review existing policies and make continuous improvements where possible based on lessons learned.

Interpreting Revenera's SCA Results for Open Source Security Vulnerabilities

Revenera uses our Code Insight product to perform SCA scans throughout the development process.

The Element of Time

Typically, analysis of reported results is constrained to reported items with associated security vulnerabilities at the time the issue was reported. It is therefore possible that there were discovered open source components with no associated security vulnerabilities that may become vulnerable in the future if new security defects are discovered. It is imperative to consider the element of time. The analysis performed is based on a snapshot of security vulnerabilities known at the time of analysis. Since open source vulnerabilities are frequently discovered and reported to various sources, the set of security vulnerabilities associated with open source components frequently changes. If an SCA scan is performed on the same codebase at a future point in time, it is very likely that the set of reported security vulnerabilities will be different from the previous scan. It is therefore critical to have an alerting mechanism in the SCA tool to notify Engineering teams if new security vulnerabilities are reported against their software bill of materials (SBOM).

Open Source Security Vulnerabilities

Security vulnerabilities can arise from both proprietary code developed by our Engineering team teams as well as from embedded open source components that are part of our applications. Open source components can be directly embedded into our application code, such as JavaScript files, or can be automatically pulled in during a build as a direct and/or transitive dependency based on manifest files that our developers maintain.

The sections below are meant to explain the various remediation options available for addressing open source security issues in applications as well as how to interpret Revenera's classifications of externally reported security issues.

Remediation Options

If a reported security issue is attributed to an open source component used by our (Revenera's) software products, the following remediation options are available to our Engineering teams:

- Revenera can **upgrade the component** to a more recent version where the security vulnerability has been fixed
- If a fix is not available, Revenera can **reach out to the author** of the open source component to request a fix for the security issue; once the fix has been made, Revenera can upgrade to the latest release of the open source component
- If a fix is not available, and the author of the open source component is not planning on making a fix, Revenera can **fix the issue**, and if possible, contribute back the fix to the author of the component so they can issue an official release; once that occurs, Revenera can upgrade to the latest release of the open source component to not be on a custom codebase
- Revenera can **swap out the component** for an alternate open source or commercial component that delivers the same or similar functionality for our application
- Revenera can **remove the open source component** from its application and implement proprietary logic for the functionality that the open source component was providing
- Revenera can **modify its proprietary code** to limit or eliminate exploitation vectors in the open source component so that the security vulnerability effectively becomes benign

Typically, the most common remediation action is to simply upgrade to the latest release of the open source component to address a security vulnerability. It is very rare for an organization's Engineering team to fix vulnerable code in open source components. Therefore, open source security vulnerabilities are rarely fixed by organizations by changing code themselves, the resolution is changing the component version or the component itself. It is important to note that not all component upgrades are straight-forward, as there are often interdependencies between components which need to be resolved to make sure that product functionality is not impacted. Often, a single component upgrade performed to address a specific security vulnerability causes additional component to have to be upgraded to maintain compatibility.

For this reason, when Revenera talks about remediation steps for reported security vulnerabilities, **we focus on the component version to which the security vulnerabilities are associated, and not the security vulnerabilities themselves**. When an offending component version is upgraded to a more recent version a list of associated security vulnerabilities that have been addressed in the code changes between versions can be generated and reported along with the component change.

Revenera's SCA data model consists of a component with a specific version and an associated list of security vulnerabilities as reported by various security repositories. The most common is

the National Vulnerability Database (NVD) maintained by the National Institute of Standards and Technology (NIST).

Revenera's Classification of Reported Issues

Typically, the following outcomes are reached based on reconciliation of reported results by an external party with Revenera's internal findings:

- **Confirmed Finding**: Revenera result match that reported by an external party
- **False Negative**: additional finding by Revenera not reported by an external party
- **False Positive**: finding reported by an external party which cannot be identified or confirmed by Revenera along with additional findings by Revenera which show up on our scans but are not valid or exploitable in the product itself

Confirmed Finding

A confirmed finding is the situation where an external party reports a security vulnerability that is traced to a specific component version that is subsequently verified by Revenera to be included in the impacted product version. In such cases, Revenera will explore the available remediation options (see above) and determine which is the most expedient fix for the issue at hand.

False Negative

A false negative finding (sometimes referred to as a Type II error) is the situation where Revenera identifies additional security vulnerabilities which were not part of the set reported by an external party. This can happen for several reasons including difference in various SCA tools, the type and depth of scan that was performed, and most importantly, when the scan was performed. The list of security vulnerabilities is dynamic and may change over time. It is common that during the investigation into reported security issues, Revenera conducts additional scans of the impacted application, and it turn, may discover additional security vulnerabilities that can be addressed as part of the remediation work to fix the reported security issues.

False Positive

A false positive finding (sometimes referred to as a Type I error) is the situation where an external party reports a security vulnerability that is traced to a specific version of an open source component which Revenera does not include in the application against which the security issue was reported. In such cases, Revenera will explain the reasoning as to why the component version at hand may be falsely reported by the SCA tool used by the external party.

Revenera will also attempt to reach consensus with the external party as to why this item is out of scope for the planned remediation work.

As part of our assessment, there will be reported security vulnerabilities due to open source components that we do not include as part of our application, but nonetheless are still discovered via Software Composition Analysis (SCA) scans. This can occur due to the following reasons:

- **Incorrect scope of analysis based on which codebase is being scanned**; typically, if you are scanning the source repo that is consumed by the build, you want to make sure that all potential dependencies are reported; however, it you are scanning build artifacts where dependencies have already been resolved, then reporting additional potential dependencies will result in false-positives since these packages are typically not included in the final application
- **Detection error by the SCA tool** (incorrect component and/or version reported)
- **Declaration of reported item in a manifest file**, but during the build this item is NOT pulled down and NOT included in the distributed/hosted application

False positives are tricky and in many cases, it may take more time to track down proof that a reported vulnerability is not exploitable than resolving a real vulnerability. This is because real issues can be tested, fixed, and verified; while a false positive ultimately requires someone to sign off on an issue that will not be addressed but continues to be reported by the SCA tool.

Below are some common scenarios based on which Revenera may consider a reported security issue as a false positive:

Incorrect Component Version

SCA tools report discovered open source components based on various automated detection techniques. In many cases, manifest files located in the scanned codebase define their direct dependencies and in turn their transitive dependencies. In such cases, results are highly accurate. However, in other cases, discovering the version for a given component is more of an art than a science, and in such cases, it is possible that a finding is less than 100% accurate. Furthermore, if the finding is not verified by an analyst, an incorrect component version may be reported.

In such a case, the associated list of security vulnerabilities will also likely be inaccurate as they are associated with a different component version than the one included in the application being scanned.

Unresolved Component Declaration

In many ecosystems, manifest files not only include declarations for direct dependencies, but also exclusions and scope options that determine when these dependencies are to be resolved by the build. Depending on how SCA scans are configured as well as the specific functionality of the SCA vendor's tool, results can be quite different.

The following scenarios can result in declared dependencies not being resolved during the build and corresponding artifacts in-turn not being part of the built application[1]:

Optional Transitive Dependency

This situation occurs when a manifest file contains a dependency declaration along with a tag which marks the declaration as optional.

An example in a Maven POM file is shown below:

```
<dependency>
  <groupId>org.cryptacular</groupId>
  <artifactId>cryptacular</artifactId>
  <version>1.2.1</version>
  <optional>true</optional>
</dependency>
```

In such a situation, our Maven build will not process this dependency and the corresponding jar file and/or class files will not be part of our application.

Test Scope Transitive Dependency

This situation occurs when a manifest file contains a dependency declaration along with a tag which marks defined the scope of the declaration as "test".

An example in a Maven POM file is shown below:

```
<dependency>
  <groupId>com.google.guava</groupId>
  <artifactId>guava</artifactId>
  <version>18.0-rc1</version>
  <scope>test</scope>
</dependency>
```

In such a situation, our Maven build will not process this dependency and the corresponding jar file and/or class files will not be part of our application.

[1] For our products developed in Java, in these situations, our Maven build will not process the these declared dependencies and the corresponding jar files and/or class files will not be part of the application.

Excluded Transitive Dependency

This situation occurs when a manifest file contains an exclusion block specifically informing the build to not process a dependency declaration.

An example in a Maven POM file is shown below:

```
<exclusion>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-core</artifactId>
</exclusion>
```

In such a situation, our Maven build will not process this dependency and the corresponding jar file and/or class files will not be part of our application.

Suppressed Transitive Dependency

This situation occurs when a direct dependency for a given component take precedence over a transitive dependency for a different version of the same component.[2]

Target Platform-Specific Transitive Dependency

This situation occurs when a dependency is declared for a specific target platform via a "provided" scope tag.

An example in a Maven POM file is shown below:

```
<!-- Provided scope: only used in Android environments - not downloaded as
a transitive dependency. -->
<dependency>
    <groupId>com.google.android</groupId>
    <artifactId>android</artifactId>
    <version>4.1.1.4</version>
    <scope>provided</scope>
</dependency>
```

In such a situation, our Maven build will not process this dependency and the corresponding jar file and/or class files will not be part of our application.

[2] For our products developed in Java, our Maven build system typically implements a "first version loaded wins for transitive dependencies" scenario. In particular, the "nearest definition" bit in the Transitive Dependencies section. Refer to http://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html for more information.

New Version of Reported Component In Use and No Evidence of Reported Version

In this situation, the reported component is confirmed, but a newer version than the one reported is actually used by our application. There is also no evidence of the older reported version of the component in the codebase. This scenario may require additional analysis of the customer's SCA scan results to determine the indicator evidence for this finding.

No Evidence of Reported Component

In this situation, the reported component is not used by Revenera's application. There is no evidence of the reported component in the codebase. This scenario may require additional analysis of the customer's SCA scan results to determine the indicator evidence for this finding.

Component Not Used by Product

There are cases where a component version is accurately reported, but for various reasons, the component is not actually used by the product and in turn the reported security vulnerability is dormant, and not exploitable.

In such a situation, it may still make sense to remove the unnecessary component depending on the overall level of effort and other competing work.

Irrelevant Vulnerability Reported

There are cases where vulnerabilities are incorrectly mapped to a component version by the SCA vendor and/or by the security repository that originally reported the vulnerability via an incorrect Common Platform Enumeration (CPE) mapping.

In such cases, while the component version is accurately reported, the corresponding security vulnerabilities are invalid and may not be a relevant finding. Usually, a security analyst can help with analyzing these results to determine the best course of action.

Vulnerability Not Exploitable

There are cases where vulnerabilities are accurately reported but based on how the application is deployed or used, the vulnerability is either not exploitable, or the severity is significantly reduced, and in turn the vulnerability fix priority is low.

In such cases a security analyst can help with analyzing these results to determine the best course of action.

How Revenera Scans for Open Source Dependencies

Revenera handles dependency scanning differently depending on whether we are scanning a source repository that the build consumes versus build artifacts, including resolved dependencies that the build produces.

In the pre-build case, when scanning a source repository, all declared dependencies, and their corresponding transitive dependencies are reported in the resulting items – the software bill of materials (SBoM). The only exception is that we skip dependencies declared with the "test" scope as these dependencies are usually not included in the final released product.

In the post-build case, when scanning build artifacts, manifest files are not scanned for dependencies since the build already processed the manifest files and pulled down the necessary dependencies into the resulting directories. Scanning the manifest files would result in over-reporting of open source components that were excluded from the build based on the combination of exclusions and scope configuration options.

Other SCA vendors do not necessarily operate the same way, and as a result, the final list of discovered items is likely to vary.

To Do in Future Version

- Explain roles and responsibilities of each team in the vulnerability lifecycle
- Develop a checklist to guide Revenera's incident response teams through the security vulnerability assessment and remediation process

Revision History

Revision	Description	Date	Author
1.0	Retitled document and added additional content about specific types of false-positive classifications, incorporated various feedback	June 17, 2021	Alex Rybak
0.3	Additional feedback	April 29, 2021	Alex Rybak
0.2	First round of feedback	April 28, 2021	Alex Rybak
0.1	Initial Version	April 27, 2021	Alex Rybak