RSAC Innovation Sandbox 2021 WINNER

Gartner COOL VENDOR 2021

The Key To Remediating Secrets in Code

apiiro Application Risk Platform
From Design to Code to Cloud

apiiro

Table of Contents

Managing Secrets in Code

Introduction	02	Secrets detection is a foundational part of your AppSec program
Secrets in Code	03	The Basics How Attackers Exploit them Identify and Manage
The Top 5 Steps	09	to Manage Secrets in Code
Take Aways	10	How Apiiro Can Help

Introduction

Software development has fundamentally changed - and security experts need to change with it. Engineers no longer write code in isolation on desktops or laptops, where an attacker compromising a device could only access locally-stored files.

Cloud-based development has transformed the security model so developers often have expanded access to the entire application. With the rise of DevOps, the same developers (and developer identities) have the ability to make changes to production environments. A single compromised identity can now have a catastrophic impact on the security of the entire application and infrastructure.

Secrets detection now needs to be a foundational part of any Application Security program.

Finding secrets in code sounds simple. Just look for field names like “password”, “token”, or “API_Key”. Maybe dig a little deeper to search for commonly-used passwords or look for randomly-generated strings of specific lengths.

Unfortunately, there is a lot of nuance and complexity to both understanding the impact of secrets in code, detecting them in the first place, and not being overwhelmed with endless false-positives but the key to managing secrets in code is context.

In this eBook, we will help you understand:

Why secrets are so commonly found in code

Types of secrets

How attackers can find secrets in your code

What attackers can do with the secrets they find

5 steps you can follow to identify and manage secrets

Secrets in Code

The Basics

Why developers put secrets in their code
It's easy to say that developers should be more careful and better follow best practices but the truth is that developers are under increasing pressure to deliver. Hard-coding a token or password may be a temporary hack before implementing a better solution later on...... that conveniently gets forgotten about as the next priority comes along.

Developers don't always have the visibility into where their code is deployed, so they don't have an end-to-end view of the risk. In addition, old code. Or old code can be deployed in new ways that were never anticipated by the original developer. It is also common to see stored secrets that were intended to never leave the development environment make their way into production.

Where secrets are found
you can find secrets in many places, including:

Source Code

Configuration Files

Infra-as-Code

Test Code

Documentation

Package Management Files

Scripts

Project Files

Secrets of each type can be in multiple environments, from staging to production. The challenge is to identify these places automatically and quantify risk for secrets in production source code vs. secrets in test code in staging and other environments.

Secrets in Code
Types of code secrets

There are many types of secrets that developers can put into code. These include:

User passwords:

The simplest type of secret is a username and password combination that is stored in plain-text.

API keys:

Application Programming Interface (API) keys can grant privileged access to key API settings.

Authentication tokens:

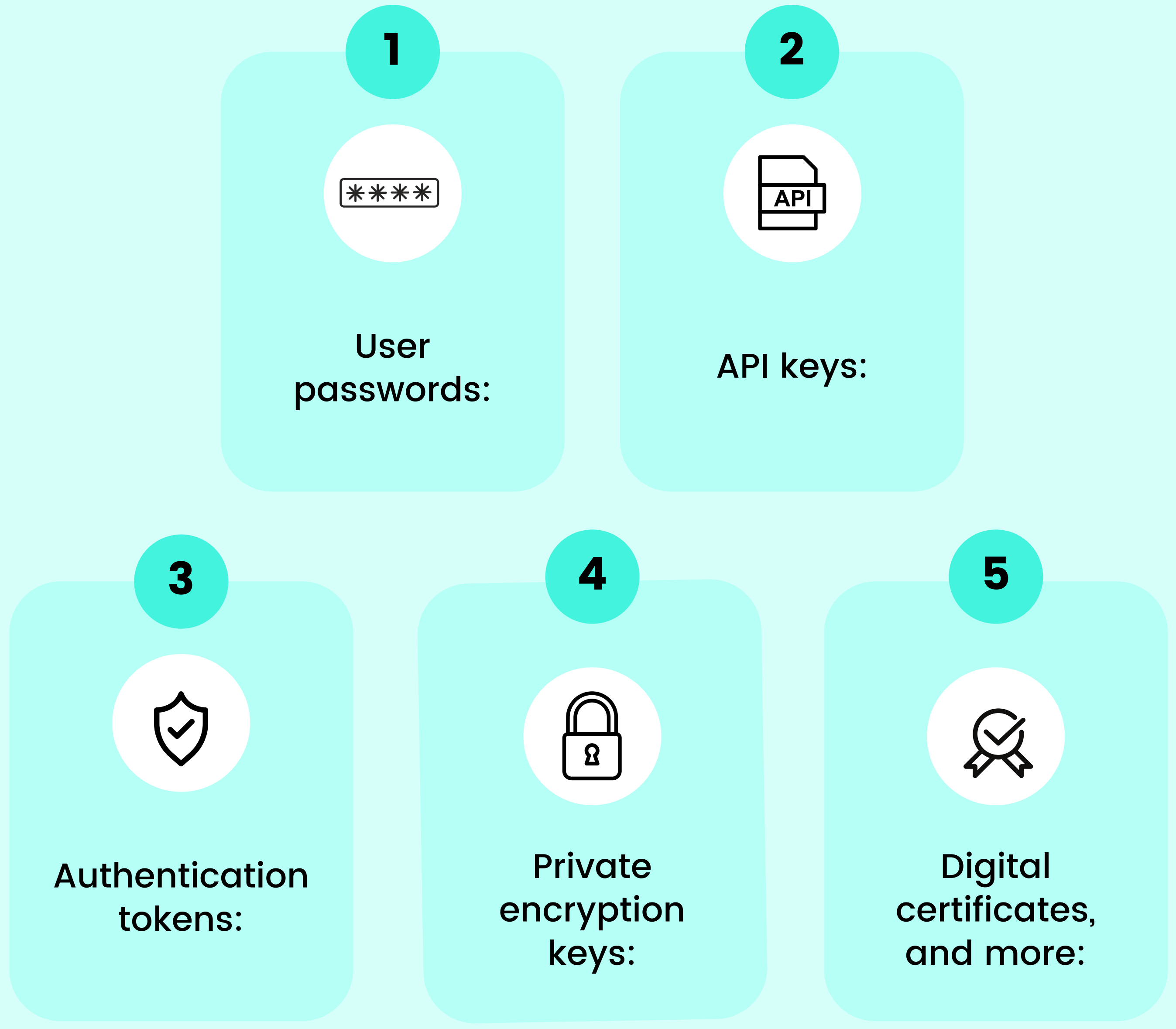
Tokens can replace less secure usernames and passwords and are used in authentication mechanisms such as OAuth.

Private encryption keys:

Private encryption keys can either be used in symmetric key encryption or can be the private half of Public Key Infrastructure (PKI) key pair.

Digital certificates, and more:

Digital certificates are used to authenticate and “prove” an identity.



Types of code secrets

Secrets in Code

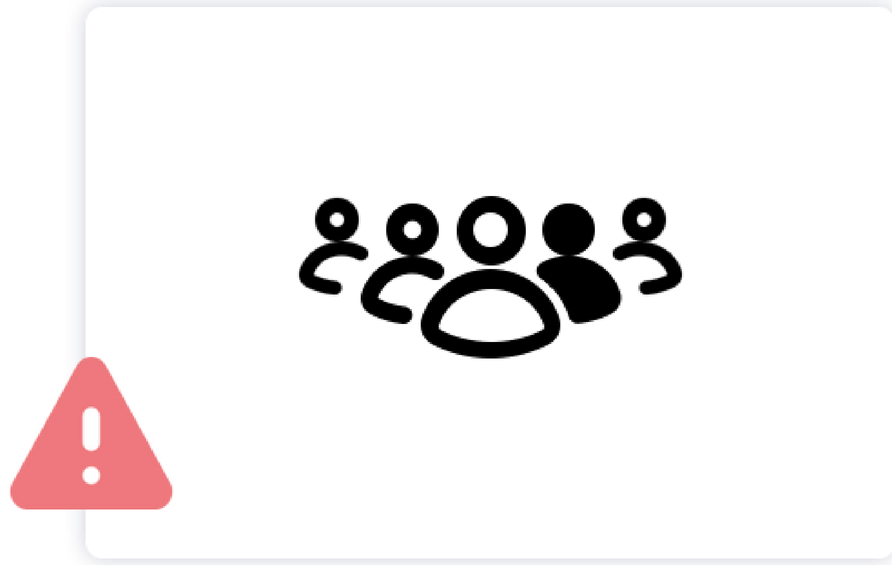
How Attackers Exploit them

How attackers get access to code secrets
Adversaries will look for secrets in any given code or data pools they can get their hands on. Low-hanging fruit will be trivially excavated from source code either from a git repository or in another form, such as client-side code (i.e., javascript processed by a browser). Even binaries are not immune to reverse engineering attacks, especially when discussing code packages that use intermediate languages, which doesn't skip popular mobile binaries.

What attackers can do with the secrets they find
The most sophisticated attacks are multi-step and a single secret can be a launch point to further command and control. If an attacker is able to find a hard-coded token, they can use it to gain whatever access that token grants. Using the right token, an attacker can impersonate a valid user or service and then use other means to escalate privileges or "jump" horizontally to other systems that use that token.

The main issue with having secrets in your code is that it short-circuits many of your defenses. For example, even if you have a SaaS product and your cloud infrastructure is secure, an attacker could use social engineering or other methods to gain access to a developer account and access the code to find and exploit the secret. And if you have 1,000 developers, all they need is access to one account!

Illustration 1
All attackers need is one developer identity



Secrets in Code

Identify & Remediate

Detecting secrets is a surprisingly complex challenge

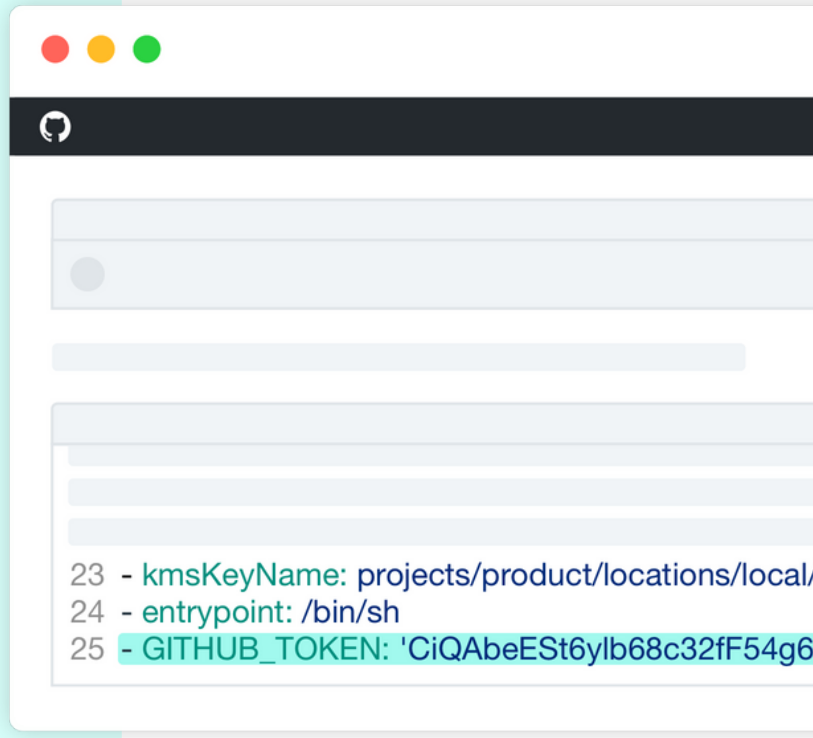
The main reason secrets detection is so difficult is that there are many types of secrets. Some values are strictly formatted for ease of recognition. For example, certificates and access tokens are generally formatted in standard ways & are easy to find, with few false positives and false negatives. Modern token formats enact several mechanisms to eliminate false positives, most prominently a checksum - a mathematical computation that checks for token validity by concatenating a trail of values at the end of the token. But not every token format features these abilities to minimize false-positives and not every tool will be tuned for utilizing it.

Other types of data are less structured and consist of long strings of random characters and may be encrypted or hashed. The problem is that you can't tell what you're looking at if you don't understand the code! If you find 100 seemingly "random" strings, some will be test files. Some will be binary files. Some will be encrypted or hashed. These are all normal to have in code and a tool should be able to distinguish them.

Cryptographic keys are usually long enough that you can use statistics to determine if the string is sufficiently random enough to be a key, but for many types of files, it isn't clear-cut and the problem becomes: how many false positives can you accept?

The other challenge has become the speed of development. Any organization can perform an ad hoc code review and identify a good number of secrets from a detailed manual search, but this isn't a scalable solution.

Illustration 2
Exposed secrets on code



```
- kmsKeyName: projects/product/locations/local/
- entrypoint: /bin/sh
- GITHUB_TOKEN: 'CiQAbeESt6ylb68c32fF54g6
```

What you can do about it

There are multiple things you can do to identify secrets, remediate issues, and reduce your risk:

Follow Key Management Best Practices.
Current security requirements call for passwords & tokens to be used only once but that's unfortunately not the reality. Developers will use API keys in test environments & the same keys for staging and production. Passwords that were always intended to be temporary get missed and left in production software in a rush to meet deadlines. Apiiro has identified keys stored in our customers' test environments and gotten an earful about overly-sensitive alerts, only to discover that those same keys were used in production. In addition, security practices need to be followed even in test environments.

Use a third-party Secrets Detection solution.
These tools will usually employ a combination of Regular Expressions (RegEx) to detect patterns in code that indicate a secret as well as entropy checks to detect randomness that may indicate a password or key and checks for checksum validity, if possible.

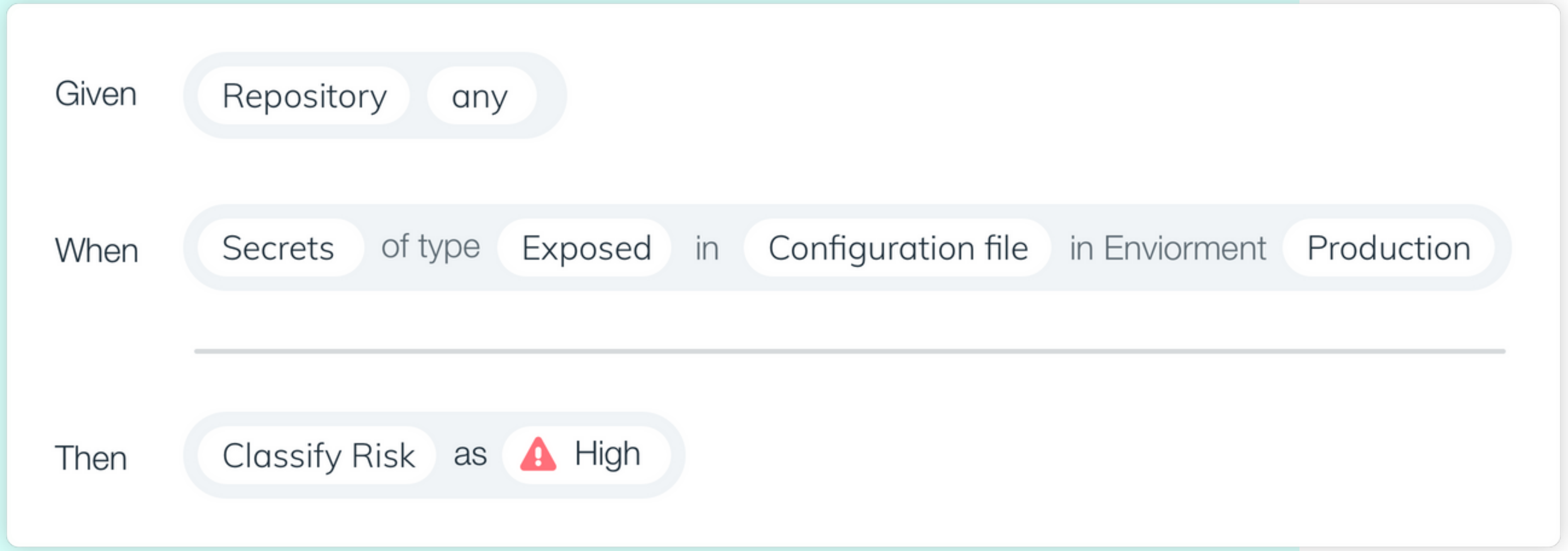

Given Repository any
When Secrets of type Exposed in Configuration file in Enviorment Production
Then Classify Risk as High

Illustration 3
Discover secrets once they are added to your source code

Secrets detection is not a stand-alone function!

Understanding and remediating the risk of secrets-in-code cannot be done in isolation. There is a significant difference in risk between finding a secret in an application with low business impact that is deployed on-premises compared to finding a similar secret in a high business impact application that stores PII! Risk is multidimensional and secrets-in-code is only one part of the larger picture surrounding multidimensional application risk.

apiiro

Where existing solutions fall short

There is an entire industry around detecting secrets in code but there are a few ways that many existing solutions fall short:

Code context.
Without a deep understanding of the code, it is difficult to minimize false positive rates. With an understanding of how the code functions, it is possible to test potential passwords, API keys, and more in a way similar to how the actual code would use them. This improves detection (and false positive rates) by orders of magnitude.

Developer Behavior.
Existing solutions do not understand individual developers in order to minimize false positives.If a particular developer has added multiple secrets to code in the past, the certainty score for secrets is higher than for developers who do not have a history of mismanaging secrets in code.

History.
Many solutions only evaluate the current source code but entirely ignore secrets that may be stored in previous versions of the application across all test environments. While secrets stored in the history of your source code manager cannot be captured by reverse engineering a binary, they still pose a risk in the event of a hacker gaining access to the source code repository, which holds all historic revisions as well.

Top 5 Steps to **Manage Secrets in Code**

Here are the top 5 phases to manage secrets in code:

Detection Identify secrets in your code using a combination of processes and tools, and aspire to have that visibility at earlier steps of the development - as early as a code commit/pull-request arrives for approval

Risk Prioritization Prioritize remediation of secrets in code according to the contextual risk. Not all secrets are the same. Carefully weigh the sensitivity of the application, the type of data stored, & the business impact of unauthorized access!

Remediation Assign a remediation task (or comment on the pull request) to the appropriate developer and alert the relevant Security Champion and Architect (via a ticketing or messaging system).

Prevention Define a workflow to automatically block pull requests and provide remediation actions the next time this type of secret is detected.

Training Provide appropriate training for the developers who mismanage secrets.

1
Detection
2
Risk Prioritization
3
Remediation
4
Prevention
5
Training

Illustration 4
The Top 5 Steps to Manage Secrets in Code

How Apiiro Can Help

Apiiro provides real-time, continuous and contextual detection of secrets, with automated workflows so you can manage your code and your risk as new secrets are introduced. Our Code Risk Platform™ automates & orchestrates secrets discovery, remediation, and prevention:

Identify Repositories
With exposed secrets across the entire history of your code.

Differentiate Secrets Type
Between exposed secrets, hashed, salted, etc.

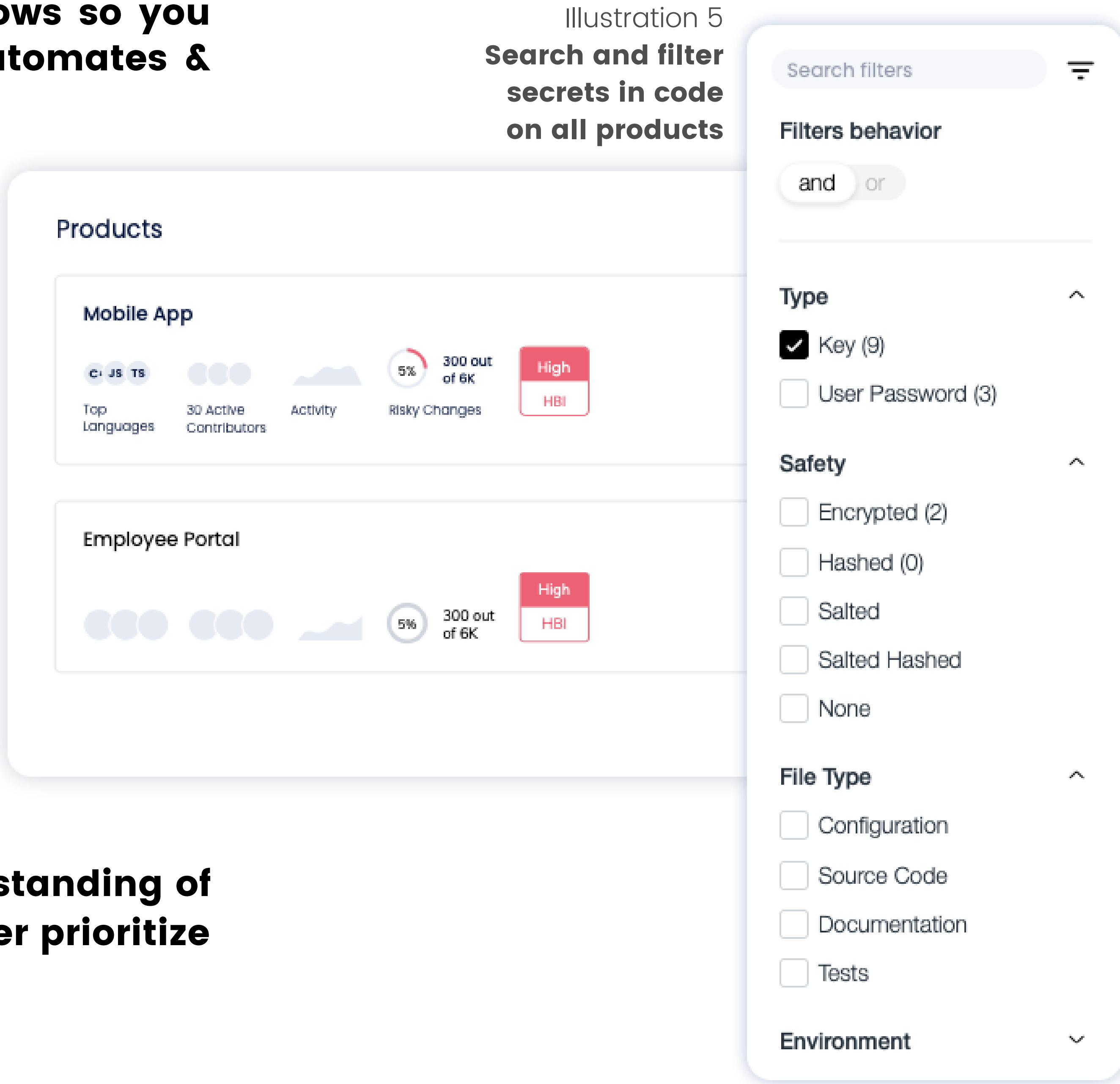
Distinguish Between Environments
Such as secrets in Test, Development, QA, and Production.

Creates automated workflows
Including comments on the pull request, send the appropriate alerts, etc. All using your existing tools, from Jira to Slack!

Provide Contextual Guidance
Understands which key management systems are already in place and instructs developers on how to remediate each issue instead of only showing alerts.

Apiiro uses the latest algorithms for entropy detection of crypto keys and leverages our deep understanding of the code to evaluate the context of each secret we find in order to lower false positive rates and better prioritize each issue by business impact.
Make secrets detection an essential part of your risk-based AppSec program with Apiiro!

Illustration 5
Search and filter secrets in code on all products



Filters behavior
[✓ and] or

Type
- [✓] Key (9)
- [ ] User Password (3)

Safety
- [ ] Encrypted (2)
- [ ] Hashed (0)
- [ ] Salted
- [ ] Salted Hashed
- [ ] None

File Type
- [ ] Configuration
- [ ] Source Code
- [ ] Documentation
- [ ] Tests

Environment

apiiro

About Apiiro

Apiiro helps you build a risk-based & measurable AppSec program. Our Application Risk Management platform provides complete risk visibility & control, from design to code to cloud. Our multidimensional approach to application security will help accelerate application delivery while reducing costs and risk.

Apiiro is re-inventing the secure development lifecycle for agile and cloud-native development, with 5 solutions:

Application Inventory & Asset Discovery Define Success with risk-based metrics that help you measure your AppSec program at both business & technical levels & Gain Risk-Based Visibility

Risk Assessment & Change Management Automate Code Governance with detailed workflows and a flexible Code Governance Engine

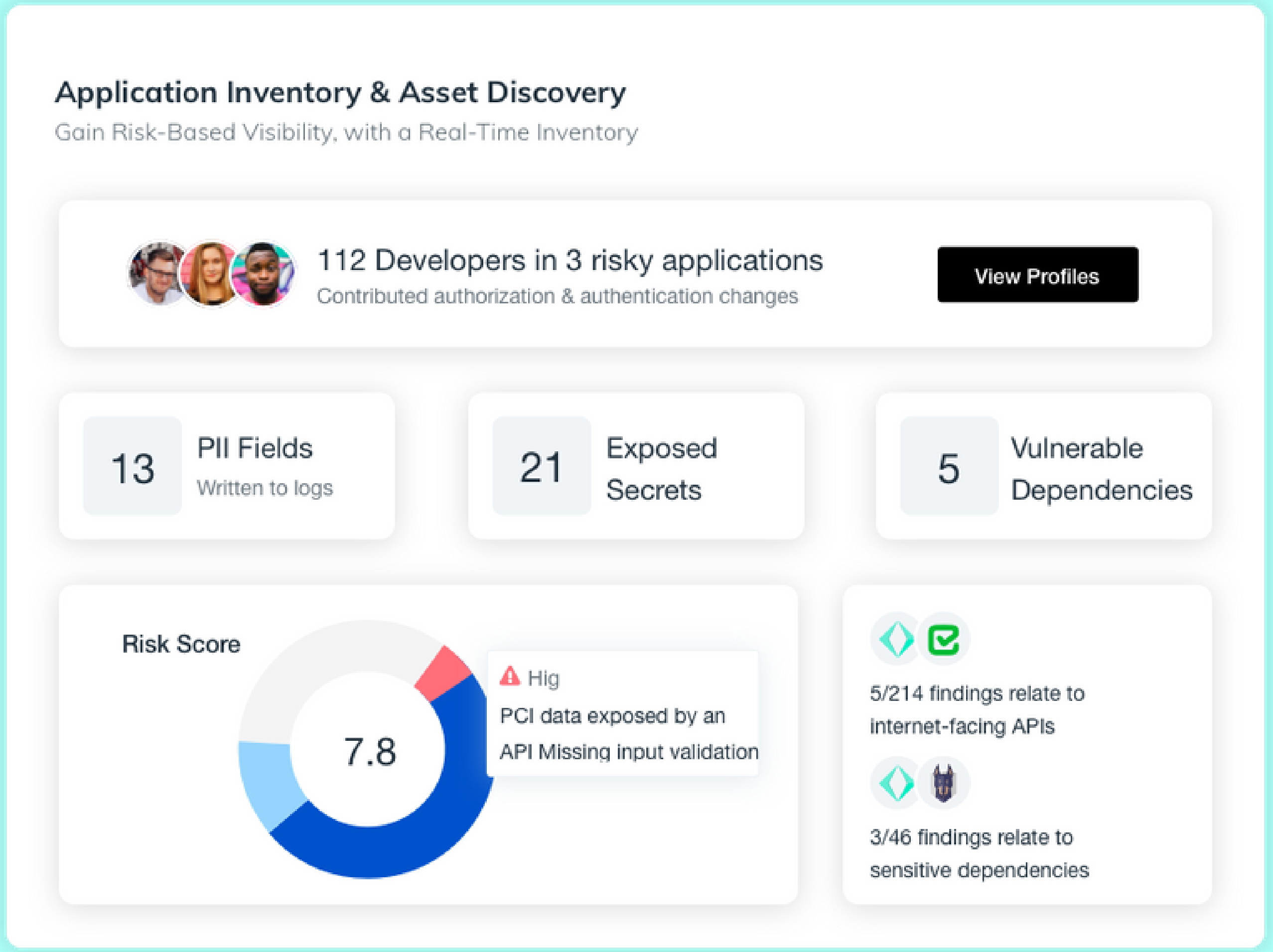
Security & Compliance Assurance Remediate Risks that Matter, with a risk-based Remediation Work Plan

Git & CI/CD Security and Integrity Shift Left & Extend Right with Security Champion identification and the context to trigger context-sensitive developer training

SSDLC Processes & Tools Orchestration Approach the SSDLC holistically with a risk dashboard that covers all SSDLC processes

Illustration 6
Apiiro solutions

Learn More →

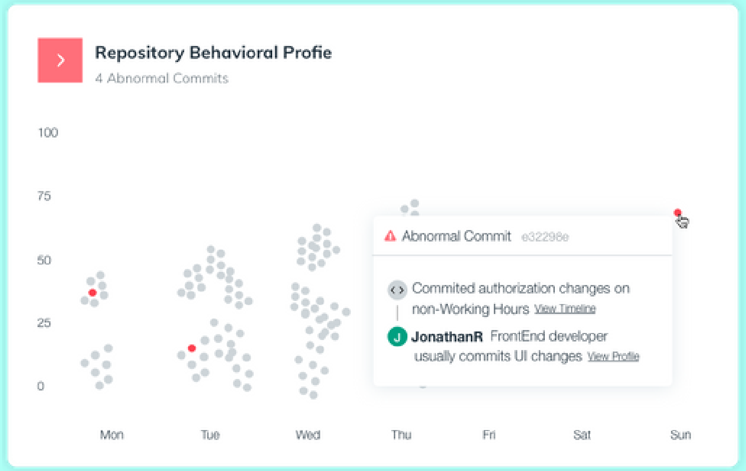

Application Inventory & Asset Discovery
Gain Risk-Based Visibility, with a Real-Time Inventory

112 Developers in 3 risky applications
Contributed authorization & authentication changes
View Profiles

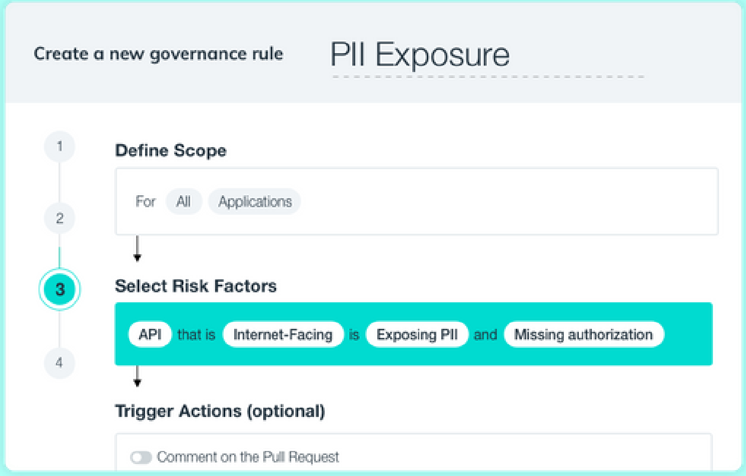
13 PII Fields Written to logs
21 Exposed Secrets
5 Vulnerable Dependencies

Risk Score 7.8
Hig
PCI data exposed by an API Missing input validation

5/214 findings relate to internet-facing APIs
3/46 findings relate to sensitive dependencies


Repository Behavioral Profie
4 Abnormal Commits

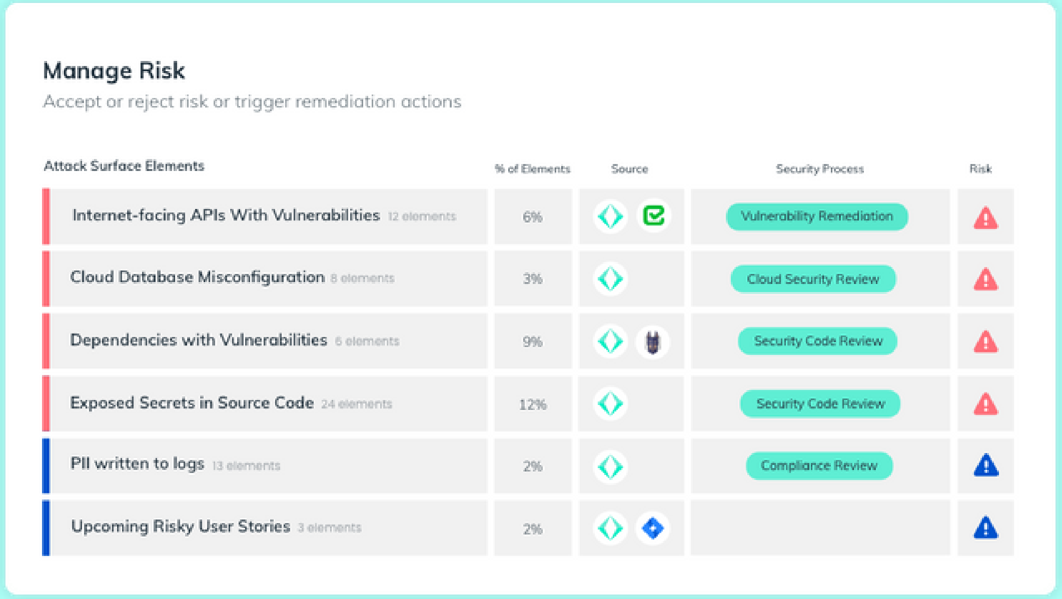
Abnormal Commit e32298e
Commited authorization changes on non-Working Hours View Timeline
JonathanR FrontEnd developer usually commits UI changes View Profile


Create a new governance rule PII Exposure

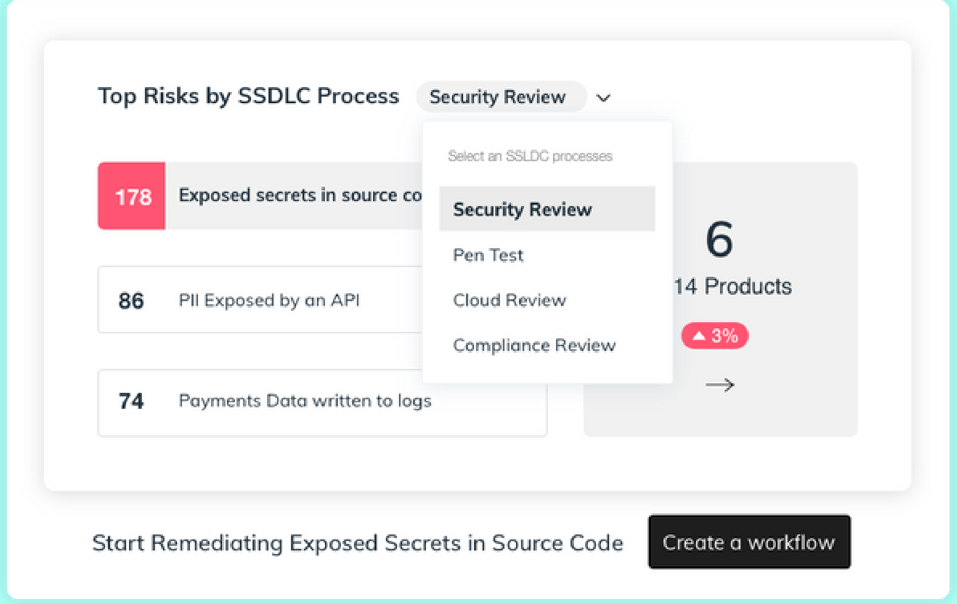
1 Define Scope
For All Applications

2
3 Select Risk Factors
API that is Internet-Facing is Exposing PII and Missing authorization

4 Trigger Actions (optional)
Comment on the Pull Request


Manage Risk
Accept or reject risk or trigger remediation actions

Attack Surface Elements	% of Elements	Source	Security Process	Risk
Internet-facing APIs With Vulnerabilities 12 elements	6%		Vulnerability Remediation	⚠
Cloud Database Misconfiguration 8 elements	3%		Cloud Security Review	⚠
Dependencies with Vulnerabilities 6 elements	9%		Security Code Review	⚠
Exposed Secrets in Source Code 24 elements	12%		Security Code Review	⚠
PII written to logs 13 elements	2%		Compliance Review	⚠
Upcoming Risky User Stories 3 elements	2%			⚠


Top Risks by SSDLC Process Security Review

178 Exposed secrets in source co
86 PII Exposed by an API
74 Payments Data written to logs

Select an SSDLC process
Security Review
Pen Test
Cloud Review
Compliance Review

6
14 Products
▲ 3%

Start Remediating Exposed Secrets in Source Code
Create a workflow