PeerSpot

Case Study

GitGuardian Internal Monitoring

Andy

Senior Security Engineer at a insurance company with 201-500 employees

Review by a Real User

Verified by PeerSpot

What is our primary use case?

We needed a detection tool that would work across all languages and help us identify problem areas. That was especially important where a codebase is made up of several different development languages written over several years (or decades).

How has it helped my organization?

GitGuardian efficiently supports a shift-left strategy. As a result, it has made things materially more secure. It's helped us to stop secrets from reaching our codebase.

The platform has helped to facilitate a better security culture within our organization. In addition to highlighting problems, it shows engineers how to properly remove them from the code, and provides advice on rotation.

The Dev in the loop feature has helped us to learn about problems and has helped us get our hands on remediating. We've gone from having very long-lived incidents to having much shorter incidents.

And because we didn't have any solution like this before, of course it has increased our secrets detection rate.

And in terms of security team productivity, using GigGuardian helped us deliver a key, strategic roadmap item for our organization.

What is most valuable?

The solution offers reliable, actionable secrets detection with a low false-positive rate. That low false-positive rate was one of the reasons we

picked it. There are always going to be some, but in reality, it's very low compared to a lot of the other, open source tools that are available.Accurate secrets detection is notoriously challenging. GitGuardian provides a rich and easy-to-use interface that enables engineers or security teams to jump on issues and manage their remediation. It offers functionality to prevent issues from creeping in.

GitGuardian has pretty broad detection capabilities. It covers all of the types of secrets that we've been interested in. For example, it covers AWS Keys. There isn't anything specific that it couldn't detect in the stack that we use. That breadth is also evident because we have a lot of different languages that it supports as well.

The "detector" concept, which identifies particular categories or types of secrets, allows an organization to tweak and tailor the configuration for things that are specific to its environment. This is highly useful if you're particularly worried about a certain type of secret and it can help focus attention, as part of early remediation efforts. The ability to check for secrets as part of pre-push hooks is fantastic, as it helps identify issues before they reach the main codebase, and that was the ultimate goal for us.

Another positive feature is that it quickly prioritizes remediation. That quick feedback loop is very helpful. Based on the detector that finds the problem, you can use that to almost rate the issue. For example, if it's an AWS Key, you would rate it very high so you can jump the prioritization accordingly, once you've got those alerts triggered. And issues can be assigned to individual developers to help gain traction on fixes.

And the Dev in the loop feature, which our developers use, is pretty important when it comes to remediation because that's what helps make the engineer responsible for having done the thing that needs remediation. This feature is effective in terms of helping collaboration between developers and our security team. It's automated, to a large extent. The "in the loop" feature will notify the engineer of what's happened and will give the security team oversight, but it deliberately puts the onus on the engineer to fix it.

In addition, the out-of-the-box reporting mechanisms allow for easy data presentation to both specific engineering teams and senior leadership.

For how long have I used the solution?

I've used the solution for one year.

What do I think about the stability of the solution?

I've had no issues with the stability of the service.

What do I think about the scalability of the solution?

I implemented it on a very large codebase, with no scalability concerns. The SaaS offering made the integration simple.

How are customer service and support?

GitGuardian's technical support is very good. They are very proactive and keen about any feedback on the detectors.

How would you rate customer service and support?

Positive

Which solution did I use previously and why did I switch?

I've previously implemented open source alternatives. These proved cumbersome, unscalable, and with such large false-positive rates as to make the output useless.

How was the initial setup?

There wasn't much preparation needed on our side to start using GitGuardian. There was just the standard opt-in to integration and we then used OKTA to manage SSO and set up integrations with GitHub. It is pretty easy.

There is no maintenance necessary because it's offered as a service.

It was a pleasure working with their implementation team to integrate it with our source control, and they were available to listen to any feedback we had.

What's my experience with pricing, setup cost, and licensing?

There are cheaper alternatives and competitors, but you get what you pay for. I've tried to implement a number of alternatives in the past, but those solutions have quickly become unmanageable due to their false-positive rates and poor interfaces.

Depending on the number of engineers committing to the codebase, pricing will very likely be a factor in any decision made. However, if you're after a great secrets detection platform, you'd be hard-pressed to beat GitGuardian.

What other advice do I have?

If a colleague in security at another company were to say to me that secrets detection is not a priority, I'd ask them why that's the case. Arguably, secrets in source code are a very large risk, especially given the distributed nature of working at the moment. Secrets detection

is pretty core for us, when it comes to application development, because we're spread out in terms of work locations. People may be using different kinds of machines to do their work, and we need to make sure that sensitive data is kept out of our codebase.

GitGuardian is a really good, well-crafted, and polished tool. You get what you pay for. It's one of the more expensive solutions, but it is very good, and the low false positive rate is a really appealing factor. And it has taught us the size of the problem that we are facing, which was something we didn't know before. It's pretty near to perfect.