PeerSpot

Case Study

GitGuardian Internal Monitoring

Danny

Chief Software Architect at a tech company with 501-1,000 employees

Review by a Real User

Verified by PeerSpot

What is our primary use case?

In general, we use Gitguardian as a safety net. We have our internal tools for validating that there is no sensitive data in there. GitGuardian is a more general and robust solution to double-check our work and make sure that if we are committing something, it only contains development IDs and not anything that is production-centric or customer-centric.

The main way in which we're using it at the moment is that it is connected through the GitHub integration. It is deployed through our code review process. When pull requests are created they connect with GitGuardian, which runs the scan before there is a review by one of our senior devs. That means we can see if there are any potential risk items before the code goes into the main branch.

How has it helped my organization?

It automates tasks and allows more individuals at the company to handle remediation. It provides visibility for the pull requests. It is integrated into our code review and deployment processes, and that integration allows the author to address an issue almost immediately, rather than waiting for a time-consuming review, and then manually asking the author to address it. It provides a nice safety mechanism, giving us some assurance that if something got forgotten along the way, we are notified before we make it a part of our codebase. It is much harder to remove something after it is merged than to do so beforehand.

It helps in quickly prioritizing remediation. We have set up GitHub and our pull requests in a way that there are numerous checks that have

to be passed. The code that is submitted can't be brought into the codebase until anything flagged is addressed as a test credential, a false positive, or the original branch is corrected. Fortunately, so far, they've all been false positives or test credentials. But it puts a stopping point in the process before it can go live with that information in there.

What is particularly helpful is that having GitGuardian show that the code failed a check enables us to automatically pass the resolution to the author. We don't have to rely on the reviewer to assign it back to him or her. Letting the authors solve their own problems before they get to the reviewer has significantly improved visibility and reduced the remediation time from multiple days to minutes or hours. Given how time-consuming code reviews can be, it saves some of our more scarce resources.

GitGuardian has also helped in bringing the responsibility of remediation to the entire team. Rather than having remediation as a part of the review process, where some of the more senior and experienced developers bring something up, it allows the whole team to handle that process. In the long run, it will encourage the team to think about those sorts of things before even submitting code, based on the responses they see from GitGuardian. It has increased the productivity of the security team by reducing the load on our small team. It puts the burden onto the entire team rather than the security team. Instead of them requesting remediation manually, it is automated as a part of our deployment process. It is definitely saving us hours per incident.

Time to remediation is now in minutes or hours, whereas it used to take days or weeks previously. That's the biggest improvement. Because it is automated and visible to the author, someone from the security team doesn't have to remind them or recheck it. That means the slowdown in the deployment process has definitely been improved by an order of magnitude. There is easily a 30-hour improvement on time to remediation, which is about an 85 percent improvement.

What is most valuable?

The Internal Monitoring is clearly the most valuable for us. We don't have a lot of public repositories, meaning the Public Monitoring is nice to have just in case something were to happen. But the Internal Monitoring catches things like IDs or tokens for some of our internal development. For that development, it's fine to have them in source control, but when those things are flagged, it is a nice reminder to the developer to double-check and make sure this is something that's only data and that there is nothing sensitive or production-related in it. In addition to being a good tool, should we have something sensitive in there, it is a nice reminder. Even though one of our senior reviewers double-checks credentials, when the developers submit something and get that warning message, they can proactively

address it.

There are a lot of nice tools, in addition to the GitHub integration, to help us as our dev team grows and to give our individual developers more responsibility, instead of just having it completely on the reviewer to validate things.

If something does pop up but perhaps the developer doesn't notice it, you can send a share link to have them review it and confirm things, such as whether it is a false positive or a test credential, and that can be done right through the share link.

The breadth of its detection capabilities is very good. There are a lot of integrations with different products, which is nice. There are some test credentials in our testing environment that are not sensitive, but it has warned us about a lot of those, although I can understand how it would consider them worth flagging. Overall, I've been impressed with what it has found. It has even found old test credentials that we don't need anymore. It has resurfaced them so that we can clean them up.

Its accuracy of detection is pretty good. The only false positives that we've had are mostly related to location, meaning closeness to a couple of the strings we use. We use a lot of unique identifiers that are 32-character-long tokens, so if they are near a word like "credential" or "password," that's the most common false positive. Configuring those as a false positive means they generally don't reoccur unless we have a new ID in there, which is pretty rare. There have been a couple of such instances, but not too many overall, given the size of our code base. At this point, we don't have those false positives because we've identified them. When we started, about 10 to 15 percent of them were false positives in that category, but after we identified them, they went away.

What needs improvement?

The main thing for me is the customization for some of the healthcare-specific identifiers that we want to validate. There should be some ability, which is coming in the near future, to have custom identifiers. Being in healthcare, we have pretty specific patterns that we need to match for PHI or PII. Having that would add a little bit extra to it.

In addition to the customization, having some kind of linking on the integration would be another improvement. The product itself is very good at grouping the same incident, but if it detected a test credential that didn't have remediation and that same one comes up in a new commit, it can be harder to find the new one. If you have a new instance of an older remediation, making sure that you're seeing the same one can be a little bit tricky. We had that issue more when we first started and hadn't gone through the original list. Now that it is cleaned up, it is less of an issue.

For how long have I used the solution?

We have been using GitGuardian Public Monitoring for about a month and the Internal Monitoring for about four to six months.

What do I think about the stability of the solution?

It seems really stable. Searches and integration are fast, and we get a response back almost immediately when making pull requests. From there, it is a matter of using the UI to find things and to send links to people. Everything has been consolidated and we haven't had any issues.

What do I think about the scalability of the solution?

So far, everything seems fast and easy. I know there is the option to build in a lot of rules, but we haven't really had to. We just let it group and do normal things, and then we just address things as they come up. There hasn't been an overabundance of false positives. It is intelligent enough to surface the right information without overwhelming us.

Currently, three people on our security team and 14 people on our dev team use it. The security team is double-checking the incidents that come in, but everyone on the dev team gets the alerts if a warning comes up during one of the pull requests. They can then sign in and address them as needed.

It is being used as part of our deployment process. I don't know how we would increase its usage. When they have the customization, we might increase usage, but that would just be another rule on the same integration.

How are customer service and technical support?

We haven't had to reach out to tech support at all. I'm optimistic, given their attention to detail on getting the integration set up and how simple it was, that it would be pretty good. But being able to figure everything out on our own has been a good sign.

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

We did not use any other solution previously. We have some pre-commit hooks that we have written that are customized for some of our own rules, but we haven't had another solution for this type of security credentials detection.

How was the initial setup?

The initial setup was very straightforward. The deployment time was five minutes. It was the easiest integration I've ever done.

We've hooked up other stuff to GitHub before,

and it usually involves a few steps. But with GitGuardian, I just generated a token and walked through it. I don't think I even read the documentation. I just found what I wanted to do, made a token, and it connected right up. I wasn't sure if I had done it correctly until I saw it started popping things in there. It was a really easy onboarding process.

Its ease of integration showed the maturity of the product or their focus in getting that process right. GitHub has its own rules and it changes a lot. Seeing how solid GitGuardian was gave us confidence in the solution.

What about the implementation team?

We implemented it on our own. For deployment and maintenance of GitGuardian, we have two people, me and one of the other admins.

What was our ROI?

We have definitely seen a return on investment. There is value in having the whole team exposed to the secrets. We do manual reviews before things get deployed, and we also run automated tests. But automated tests can take a while to run, while this runs pretty quickly. Having that feedback so that something gets detected before the review starts really saves a lot of time for some of our more senior and busier devs who are doing manual reviews. That time saved gives us ROI. Rather than starting a review and then having to do a new review after the secrets have been addressed, they are now able to ensure that all secrets are addressed before they review something.

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

Its pricing is very reasonable for what it is. We don't have a huge number of users, but its yearly rate was quite reasonable when compared to other per-seat solutions that we looked at. I'm not aware of any costs in addition to the standard licensing fees.

Having a free plan for a small number of users was really great. If you're a small team, I don't see why you wouldn't want to get started with it.

Which other solutions did I evaluate?

We looked at a couple of other solutions. GitGuardian seemed to be the most robust. It had different ways to connect and validate the code. We wanted to see it with our code and the pull requests. The ease of connecting the integration was definitely a major positive. We were able to integrate it quickly and easily and see the results right away. It checked off the requirements we had. It also integrated with a lot of different things, and it had a lot of robustness not only around secrets detection

but also around how they were handled.

Seeing how quickly it could produce search results on the public side, and knowing how much is in GitHub that is public, was really impressive. We knew it wasn't going to be a burden on our deployment process or that we would be waiting for it a lot. Once it was hooked up, its speed and accuracy made it a pretty easy decision to get it.

The other solution that was in the running felt like a very new product, and there was a lot more manual customization to get it to be as clear and as well-categorized as GitGuardian. That other solution was a centralized place and more automated than our process was, but it wasn't as well thought out and as well organized as GitGuardian. We got a lot more out-of-the-box with GitGuardian than we would have gotten with the other solution. Given that it is for secrets detection, you have to have confidence in the solution you go with. The other solution not being a robust solution was something of a red flag for us. We wanted something that was very well thought out from the beginning, because of the sensitive nature of what it is doing.

What other advice do I have?

I would advise others to give it a try. It is easy enough to integrate with your process, and you'll see the value right away, with a couple of quick test scenarios. Once you see it in action, it sells itself.

If a colleague at another company said to me that secrets detection is not a priority, I would ask what is more of a priority, and then I would point to a quick Google search with a myriad of issues and data breaches that have happened from leaked secrets. That is pretty easy to find. If leaks are happening, and there is a reasonable plan, or even a free plan for a small number of users, to deal with them, I don't know how much more bang for your buck you can get. I would tell him to consider the small amount that GitGuardian costs and the value and ease of integration that it provides.

Secrets detection is extremely important to a security program for application development, especially on a team of people with various experience levels. Having something automated always improves things. Having that detection on top of any of your manual processes adds an extra layer of safety. Given the ease of integration, it is extremely important and extremely valuable to have that extra layer of protection to warn you if you do forget something.

So far, GitGuardian hasn't detected any true secrets in our code. They were only internal credentials, but it has certainly brought a much-needed discussion about those test credentials. Fortunately, we've been successful at not committing production secrets since we started using this solution.

The biggest lesson that I've learned from using this solution might not be so much from secret detections, per se. It is about the ease of

integration and what going the extra mile actually does. It creates a positive experience, and it also helps in creating a lot of faith in the solution, overall. With the onboarding experience being handled very well, it gave me a lot of confidence that this was the right solution. That's a lesson for our own software. It is super important to have that ease of getting started. That can go a lot farther than you might think for the effort it requires in the overall project. I'm sure a lot more resources are spent on the analysis and the tool itself, but don't skimp on the onboarding.

I would rate GitGuardian a nine out of 10. The two areas for improvement are probably the only things that are keeping me from giving it a 10. The major one of those is probably going to be addressed pretty soon. Once we can do some of those custom identifiers or custom rules, it would be a 10.

Read 5 reviews of GitGuardian Internal Monitoring

See All Reviews