



The Top 6

Unknown SDLC Risks Legit Uncovers



When security teams first start using the Legit Application Security Posture Management (ASPM) platform, the most common reaction is surprise at the amount of unknown risk lurking in their software development environments. Since so many of these risks are commonplace across enterprises, we thought it would be beneficial to share our top findings, along with tips and advice on avoiding them.

In our work with enterprises in industries from financial services to healthcare, high tech and more, we most often uncover:

01

Exposed secrets

[Learn more →](#)

02

Unknown build assets

[Learn more →](#)

03

Misconfiguration of build assets

[Learn more →](#)

04

Developer permissions sprawl

[Learn more →](#)

05

Missing AI guardrails

[Learn more →](#)

06

IaC misconfigurations

[Learn more →](#)

01 Exposed secrets

What we find

Secrets are extremely pervasive in software development environments, and their exposure is one of the most common risks unearthed by the Legit platform. This is troubling because secrets are often the first foothold that attackers leverage to mount much larger attacks.

The types of exposed secrets we find most often include:

- Cloud keys (AWS/GCP)
- GitHub personal access tokens and CI/CD server keys (Jenkins, ADO)
- PII, such as Social Security and credit card numbers

The Legit research team finds an average of 12 secrets submitted per 100 repositories every week.

And exposed secrets are not just a hypothetical risk. In a [recent survey](#) of 350 IT and cybersecurity professionals and application developers, TechTarget's Enterprise Strategy Group (ESG) found that the top cybersecurity incident (related to internally developed cloud-native apps) experienced by survey respondents in the previous 12 months was secrets stolen from a source code repository.

The screenshot shows a detailed alert for a 'Public secret detected: Google API Key'. The alert is categorized as 'Critical' and 'Secrets'. It indicates that a possible Google API key was found in a public repository, which is considered compromised. The alert includes a score of 100, a validity status of 'Valid' (+20 score points), and a base severity of 'Critical' (+80 score points). The origin is identified as the repository 'ZatrosSecurity/youtube-di', which is public. The secret was committed by 'jdoe@gmail.com' on 10/23/2020 at 09:31 AM. Two examples of the secret are shown: a search key in a URL and a query parameter in a JSON object. The alert also provides remediation steps, such as revoking the key and using a password manager or environment variable. The alert is managed by 'Jordan Doe' and has 'Jira options' available for closing.

Legit secrets detection and remediation

Where do we find exposed secrets?

We regularly find exposed secrets in source code, which can be accessed by any user with access to the repository.

But increasingly, we are finding exposed secrets in many other places as well — like yaml files, build logs, containers, bash scripts, artifacts, Jira, Confluence, Slack, and more.

Why is secrets exposure pervasive?

Why has secrets sprawl become such a significant issue? Partly because modern development requires lots of different tool sets that need to integrate with each other, and modern apps require hundreds of secrets to function (API keys, third-party tokens, cloud credentials, etc.).

At the same time, developers are pushed to innovate and develop code as fast as possible, frequently leading to shortcuts intended to drive efficiency and speed. One of those shortcuts is using secrets in development to accelerate testing and QA.

The problem is that it's very easy for these secrets to remain exposed. For example, a developer may test a piece of code with a key. When it works, they move it into production without removing that key. They either forget, or the key works and they don't want to adjust it.

This practice and others like it lead to a continuously growing and significant source of risk to the organization.

🎯 Related attacks

Sisense

APRIL 2024

A significant [data breach at Sisense](#) was traced back to an accidental exposure of sensitive data via a GitLab repository. A hardcoded secret in the repository provided unauthorized users access to Sisense's cloud storage, leading to the compromise of vast amounts of customer data.

Toyota

OCTOBER 2022

[Toyota announced a data breach](#) caused by a subcontractor who accidentally published source code containing a hardcoded secret access key on a public GitHub repository. This exposed the personal data of nearly 300,000 customers, highlighting the risks associated with insufficient security practices around sensitive information management.



What we recommend

To prevent exposed secrets

Focus first on SaaS services keys (e.g., AWS access keys), since if code is leaked, credentials to SaaS services are immediately usable if they are valid, whereas internal credentials require attackers to also have network connectivity.



Related Resources

Secrets security

[Learn more →](#)

Our recommended best practices include:

-
- Avoid committing secrets to any Git repository. Once in the Git history, remediation steps are lengthy.
 - Avoid git add * commands.
 - Name sensitive files in .gitignore.
 - Don't rely on code reviews to discover secrets.
 - Use automated secrets scanning on repositories.
 - Use CLI or pre-hook commit tools when able to catch secrets before they get to your Git repository.
-
- Change the source code to not rely on hard-coded secrets by using a password manager, environment variable, etc. Then revoke the sensitive data.
 - Use encryption to store secrets within Git repositories.
 - Use local environment variables, when feasible.
 - Use secrets as a service solutions (e.g., Hashicorp Vault, CyberArk Conjure, etc.).
-
- Avoid secrets within build logs and sharing secrets via messaging services.
-
- Reduce AuthZ and Admin credentials to least privileged.
-

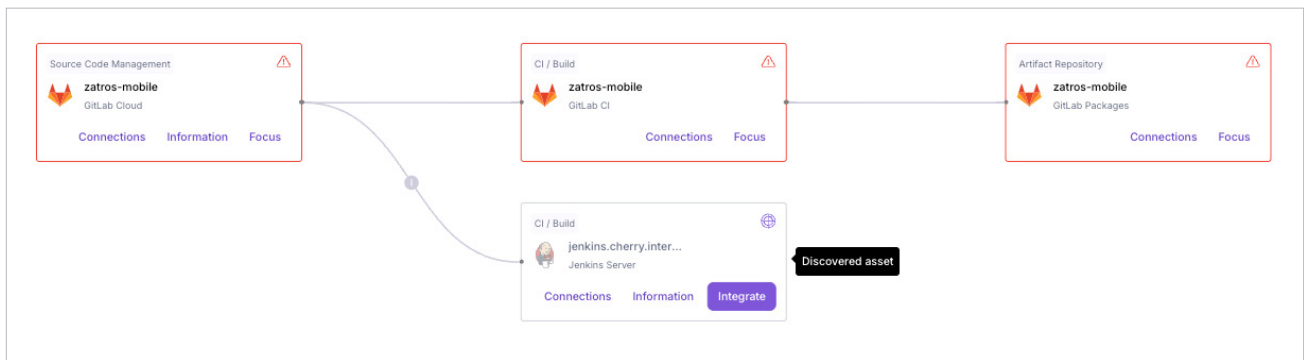
02 Unknown build assets

What we find

We commonly uncover build assets in organizations' development environments that they are not currently using, and aren't aware of. For example, working with a major manufacturing enterprise, we exposed a rogue Jenkins server in their environment; they weren't aware it was there, and it was not only exposed to the Internet, but also had risky misconfigurations.

Why are there so many unknown assets in build environments today? Developers "bringing their own devices" plays a big role.

In the not-so-distant past, there was a sharp divide between dev and ops — developers created applications, and operations ran them. Now development teams are creating their own production and build pipelines. This helps development move substantially faster, but security has lost visibility. We often come across scenarios where development teams are spinning up Jenkins servers, Azure DevOps pipelines, or JFrog artifact repositories, and the security team has no visibility into where or when.



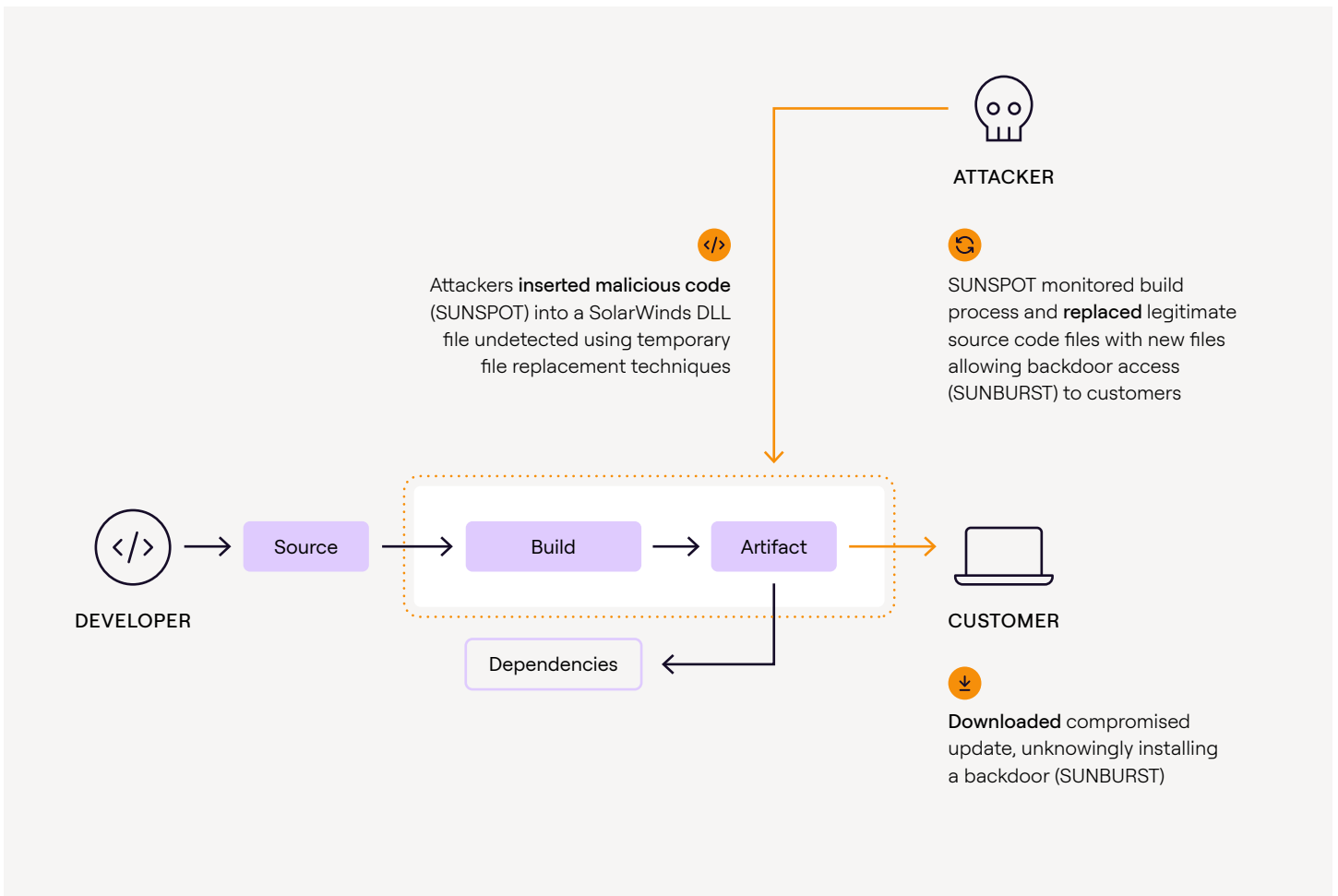
Legit identification of unknown assets


🎯 Related attacks

SolarWinds

The SolarWinds attack, where hackers deployed malicious code into its Orion IT monitoring and management software used by thousands of enterprises and government agencies worldwide, originated with an unknown asset.

The attacker entry point was an unknown, and un-monitored, build server. Misconfigurations played a role as well, but the lack of visibility into build assets was the first flaw in a series of vulnerabilities.



 What we recommend

To prevent unknown build assets

Trust but verify. Most development teams are happy to share what's in use where, but sometimes there are assets the development team doesn't even know exist. And consider situations like M&A where a flood of new assets are suddenly in play.

 Related Resources

Securing build systems

[Learn more →](#)

In the end, you need a tool or process that will help you answer the following:

-
- Do you know where all of your developer assets are?

 - Can you easily map out your pipelines from start to finish?

 - Can you tell when developers spin up a new Jenkins environment and things like that?

 - Do you know where you have appropriate security controls within your pipeline?

03 Misconfiguration of build assets

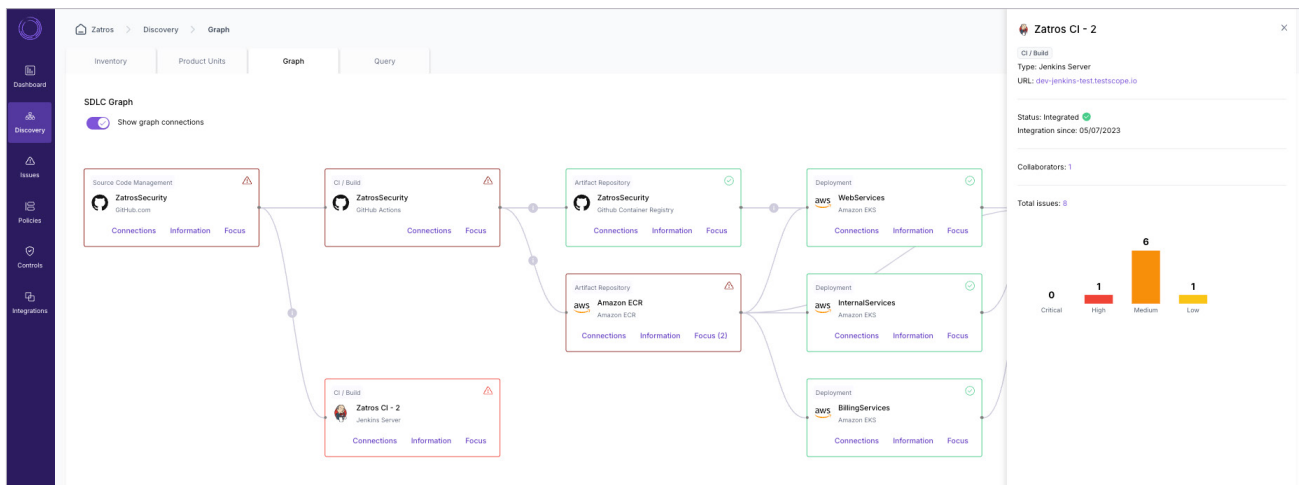
What we find

Misconfigured SDLC assets, such as SCMs, build servers, and artifact repositories, provide an opportunity for threat actors to gain access to systems and then move laterally within an organization. These assets all provide configuration mechanisms to prevent this, but they need to be used and continually monitored for proper use.

That use and monitoring are clearly not consistent, as we frequently find misconfigured build servers. This is a common problem, but also one that creates significant vulnerabilities. Build systems are essentially automated, implicitly trusted pathways straight to the cloud, yet most aren't treated as critical from a security perspective. In many cases, these systems — like Jenkins, for example — are misconfigured or otherwise vulnerable and unpatched.

Working with one large enterprise, the Legit team found an exposed Jenkins server with access to the public Internet. The Legit research team was able to access proprietary code via the Internet through that Jenkins server.

We also often find Jenkins servers that have unnecessary access to many S3 buckets. An attacker who breaches a server with this kind of access has a treasure trove of data to pursue.



Legit identification of misconfigured build asset

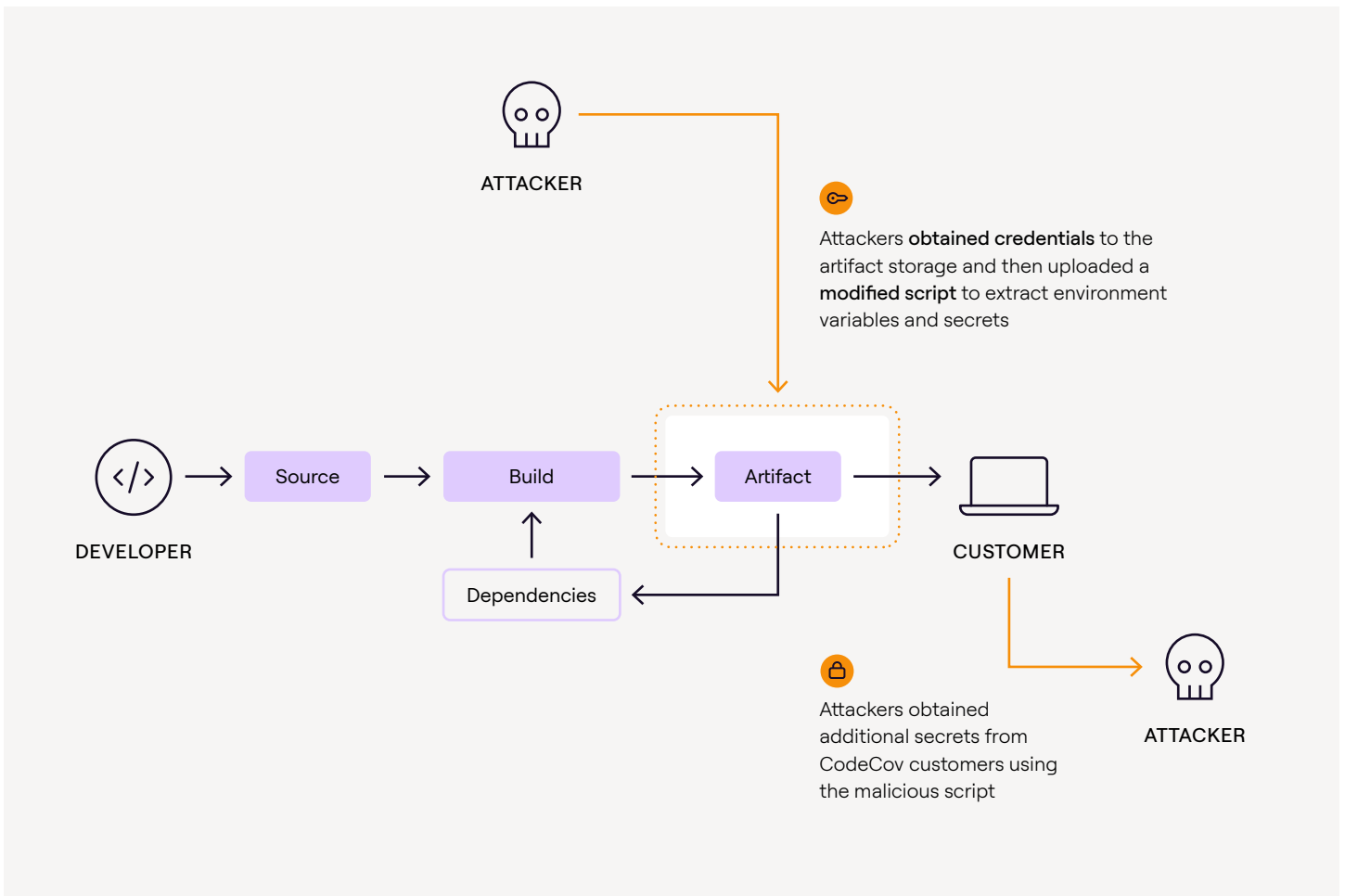
🎯 Related attacks

Codecov

In the [Codecov attack](#) (shown below), attackers used an unpinned Docker image to alter the Bash Uploader script. This modification enabled them to steal sensitive data from many of Codecov's clients, highlighting the risks of not locking Docker image versions in CI/CD pipelines.

Kaseya

[Attackers used an exploit](#) to get into a build pipeline and change code in a package that was then sent to all of Kaseya's customers.



To prevent misconfigured build assets

→ Branch protection

Enable branch protection and enforce code review (where the reviewer is not the committer) for all important repositories.

→ Continual monitoring

Keep in mind that it's important to have continual monitoring and verification of configurations. We recently partnered with a security team at a large enterprise; the team would enable branch protection, and then their development team would disable it. Obviously, developer/security communication and collaboration goes a long way here, but also continual monitoring of a constantly changing environment.

→ Enforce authentication

Ensure build servers require authentication. Some build servers, like Jenkins, have configuration settings that do not require authentication, which allows any user with network access to the server to perform any action.

→ Expire keys

Security keys are often not set to expire by default. Ensure that your infrastructure settings define security key expiration times and key rotation.

→ Limit permissions

Limit the ability to create public repositories. When non-admins can create public repos, it increases the likelihood that a repo that should be private becomes public by mistake, and once public, it can be published, cached, or stored by external parties.

Similarly, ensure that cloud storage (e.g., S3 buckets) is not publicly writable and ensure it is only publicly readable when necessary. Unprotected S3 buckets are one of the major causes of data theft and intrusions.

→ Never execute third-party resources before verification

Third-party resources should be verified by checksum or, if checksum is unavailable from the supplier, consumed from the local artifact registry after it's been downloaded and reviewed. If users of CodeCov followed this best practice, they would have caught that the checksums did not match and avoided being collateral damage from their compromised supplier.

→ Avoid unsafe cross workflow actions

When possible, avoid creating a job in your builds that references another image that might be changed externally. For example, avoid always pulling the "latest" image from an external source because if that image is compromised, you will automatically pull in the affected image.

04 Missing AI guardrails

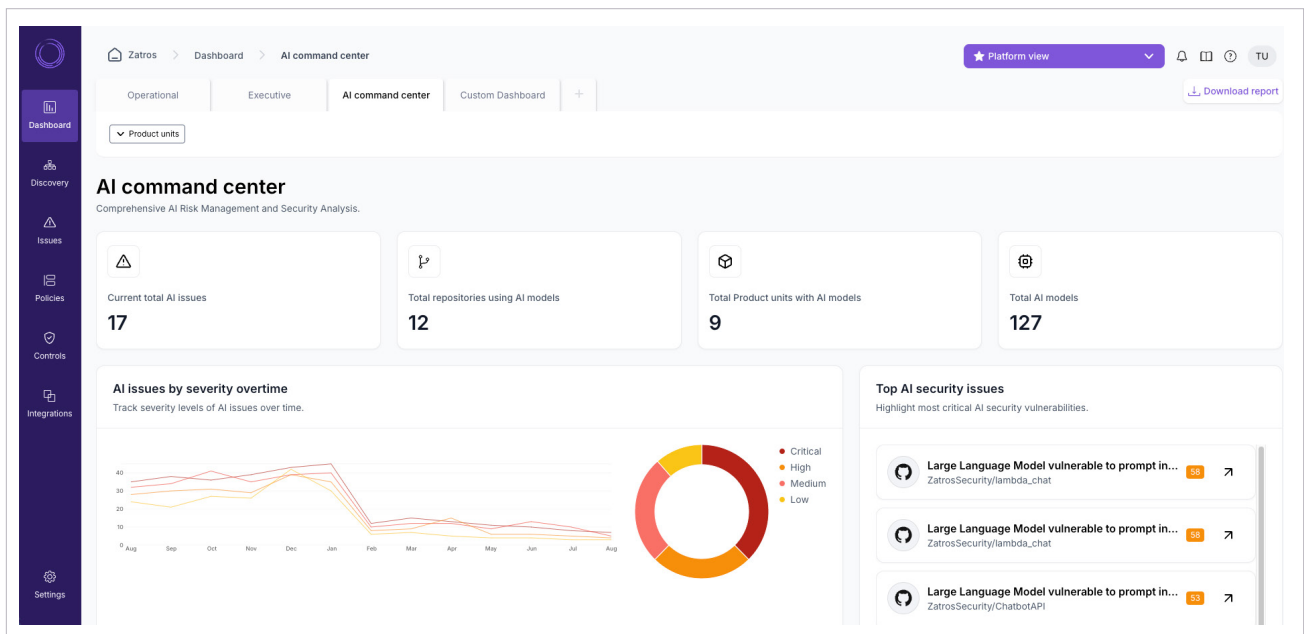
What we find

GenAI has recently emerged as an additional unknown risk we uncover. Although it gives developers an easier way to produce code at scale, it also adds risk.

We often discover that security teams first don't know where AI is in use, and then find out it's used in a location that isn't configured securely. For instance, a developer is using AI and generating code on a repository that doesn't have a code review step.

This could, for instance, allow for licensed code to enter the product, exposing the organization to legal or copyright issues.

We also often detect low-reputation LLMs in use, which could contain malicious code or payloads, or exfiltrate data sent to them.



Legit AI security command center

🎯 Related threats

We're already seeing just how vulnerable organizations are to AI models when they aren't properly monitored, secured, or managed. Developers are mistakenly leveraging malicious AI models available on open-source registries (e.g., Hugging Face) in their own software projects. And even more LLMs and AI models contain bugs and vulnerabilities that have the potential to cause AI supply chain attacks, like the vulnerabilities Legit recently uncovered in LLM automation tools and vector databases or the AI Jacking vulnerability Legit discovered last year.

Every day, there are more reports of AI security vulnerabilities from prompt injection to inadvertent data disclosure to poor implementations and misconfigurations of LLMs in applications.

And AI security risks go well beyond open-source AI. Leading providers of commercial and proprietary AI products have experienced their fair share of security setbacks themselves. For instance, OpenAI disclosed a vulnerability last year in ChatGPT's information collection capabilities that attackers could exploit to obtain customers' secret keys and root passwords.

This risk is compounded by the fact that security teams typically have very little visibility into where and how GenAI is being used by development teams.

✦ What we recommend

To prevent AI risk

Security teams need to know (and typically don't) who's using AI, where they are using it, and if there are guardrails in place in those areas.

Best practices include:

- Threat modeling the impact of AI-specific threats
- Beyond functionality and performance, considering security when selecting AI models
- Employing tools to get visibility into AI use across your development environment
- Protecting AI models from direct or indirect access

05

Developer permissions sprawl

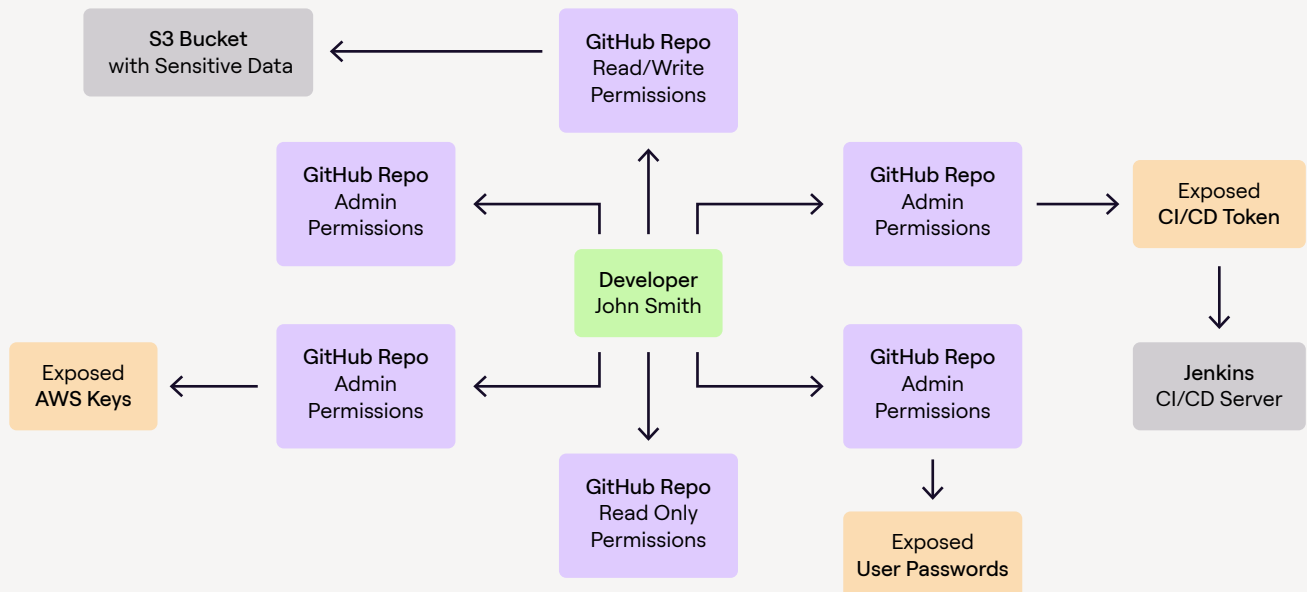
What we find

Developer permissions sprawl is a widespread issue. We often see organizations giving developers admin access to every repo by default during onboarding. With this setup, if one developer's credentials are compromised, an attacker now has access to everything.

Keep in mind that a wide variety of people have permissions to access the source code management systems, CI/CD systems, and artifact registries that make up your software development processes. Each of these systems has a different permission model, but all are orchestrated together, and the complex infrastructure that underpins the SDLC

makes it difficult to manage all the permissions a user may have. Add to that the fact that organizations often have a bad security habit of sharing credentials across systems, and you now have most of your development team with powerful access to a large portion of your company's SDLC.

This complex and untamed web of permissions creates an easy target for attackers. All they need to do is gain access to the right developer or account — either active or dormant — and they could obtain broad and powerful access to your build environment where they can wreak havoc, stealing IP and inflicting downstream damage to your customers.



🎯 Related attack

LastPass

The [LastPass attack](#) was a good reminder that it only takes one compromised account for a malicious actor to gain access to the entire SDLC.

✦ What we recommend

To prevent permissions sprawl

→ Establish policies for permissions and then regularly audit permissions.

→ Additionally, consider using RBAC for provisioning permissions so onboarding and offboarding is more scalable.

06 IaC misconfigurations

What we find

One increasingly common risk we identify is infrastructure as code (IaC) misconfigurations.

Infrastructure as code use is exploding as developers look for ways to move faster. With IaC, developers can provision their own infrastructure without waiting for IT or operations. However, with increased use comes increased chance of misconfigurations. In fact, 67% of survey respondents in a recent [ESG survey](#) noted that they are experiencing an increase in IaC template misconfigurations. These misconfigurations are especially dangerous because one flaw can proliferate easily and widely.

Examples of these misconfigurations include:

- Not using the latest version of TLS
- Relying on username/passwords instead of SSH keys
- Using vendor-supplied keys as opposed to CMKs

Name	Policy	Detection date	ID	Asset	Tickets	Legit assignee
Ensure all data stored in Aurora is securely encrypted at rest	Ensure all data stored in Aurora	12/18/2023 12:40 PM	68E0736F	ZatrosSecurityHerragot
Ensure all data stored in Aurora is securely encrypted at rest	Ensure all data stored in Aurora	12/18/2023 12:40 PM	9CF3908CE	ZatrosSecurityHerragot
Ensure all data stored in Aurora is securely encrypted at rest	Ensure all data stored in Aurora	12/18/2023 12:40 PM	85C0558E	ZatrosSecurityHerragot
Ensure all data stored in Aurora is securely encrypted at rest	Ensure all data stored in Aurora	12/18/2023 12:40 PM	EF2627M7	ZatrosSecurityHerragot
Ensure all data stored in Aurora is securely encrypted at rest	Ensure all data stored in Aurora	12/18/2023 12:40 PM	F48C0A9A8	ZatrosSecurityHerragot
Ensure all data stored in Aurora is securely encrypted at rest	Ensure all data stored in Aurora	12/18/2023 12:40 PM	A597A8F1	ZatrosSecurityHerragot
Ensure all data stored in Aurora is securely encrypted at rest	Ensure all data stored in Aurora	12/18/2023 12:40 PM	CA74C0C48	ZatrosSecurityHerragot
Ensure all data stored in Aurora is securely encrypted at rest	Ensure all data stored in Aurora	12/18/2023 12:40 PM	2A378A233	ZatrosSecurityHerragot
Ensure all data stored in Aurora is securely encrypted at rest	Ensure all data stored in Aurora	12/18/2023 12:40 PM	06C08236D	ZatrosSecurityHerragot

```
129 resource "aws_rds_cluster" "app9-rds-cluster" {
130   cluster_identifier = "app9-rds-cluster"
131   allocated_storage = 10
132   backup_retention_period = 25
133   tags = {
134     git_commit = "8791e74553b6d6887c245664f8d8cf676c92f28e5"
135     git_file = "terraform/ha/rds.tf"
136     git_last_modified_at = "2021-12-08 23:26:32"
137     git_last_modified_by = "testuser@gmail.com"
138     git_modifiers = "testuser"
139     git_org = "test"
140     git_repo = "terragoat"
141     yrk_trace = "84C98338-c751-4743-92f1-a186ca758249"
142   }
143 }
```

Legit identification of IaC misconfigurations

🎯 Related attack

A [misconfigured Amazon S3 bucket](#) resulted in 3TB of airport data being publicly accessible, including airport worker ID photos and other PII.

✦ What we recommend

To prevent IaC misconfigurations

→ Inspect all infrastructure as code findings with an eye for ensuring the most secure options are used wherever possible.

→ Make sure to use a reputable scanner, such as Checkov, to detect issues.

→ Always use known, reputable templates so that developers have a secure baseline to work from when creating new IaC artifacts.

→ Pay special attention to access controls in IaC definitions:

- Determine what users need to do, then craft policies allowing them to perform only those tasks.
 - Do not allow all users full administrative privileges.
 - Start with a minimum set of permissions and grant additional permissions as necessary.
-

Know the unknowns

Visibility has become a core requirement for software security. Development is so fast-moving, complex, and independent that getting a handle on what you have where is now a necessary first step.

→ Learn more about ASPM and the problems Legit is solving.

[Learn more](#)

→ Better understand how we can help your organization.

[Request a demo](#)



Legit is a new way to manage your application security posture for security, product and compliance teams. With Legit, enterprises get a cleaner, easier way to manage and scale application security, and address risks from code to cloud. Built for the modern SDLC, Legit tackles the toughest problems facing security teams, including GenAI usage, proliferation of secrets and an uncontrolled dev environment. Fast to implement and easy to use, Legit lets security teams protect their software factory from end to end, gives developers guardrails that let them do their best work safely, and delivers metrics that prove the success of the security program. This new approach means teams can control risk across the business — and prove it.

legitsecurity.com