

# A New Approach to Application Security

*Stop Collecting Tools, Start Building a Foundation*



**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.

**[Book a demo today!](#)**

# Contents

2

---

Modern Software Development:  
More Speed, More Sources, More Risk

---

6

---

Security Challenges Introduced by  
Modern Software Development

---

13

---

How Attackers Are Taking Advantage

---

16

---

Shortcomings of the Current Application Security Approach

---

20

---

A New Approach to Application Security: Visibility, Prioritization, and Alignment

---

27

---

Don't Install Security Cameras Before You Know the Layout of the House




---

# Modern Software Development: **More Speed, More Sources, More Risk**

.....

From multi-cloud deployments to microservices, containerization, artificial intelligence, automation, CI/CD pipelines and DevOps, software development has changed dramatically in recent years and is now faster, more automated, more dynamic, and highly reliant on third parties.

# Then and Now

	Consider the not-so-distant <i>past</i>	Fast forward to <i>today</i>
<b>Architecture</b> 	Software had monolithic architectures.	Most software is cloud native and built with a modern microservice architecture.
<b>Updates</b> 	It was updated quarterly, semi-annually, or even annually.	Deployments happen daily, sometimes even thousands of times per day.
<b>Development</b> 	There was a very real divide between the way developers created, tested, and rolled out code for applications, and the hardware or virtual machines used to host the code.	Everything can be built with code, not just the application itself, but the infrastructure, the build pipelines, the automation testing, and the runtime environments.

Past

Developers created code, then it went to a completely different team to deploy and operationalize. If they got stuck on a piece of code, they would post on Stack Overflow, ask another senior developer, or research a solution on their own — all of which could take days.

Today

Some organizations have even adopted development practices that split teams into pods that develop, deploy, and maintain their own services from code to cloud.

Developers are now more self-sufficient and capable than ever before. They can build not only their own apps, but also local test environments that look identical to production environments and pipelines for continuous integration and deployment (CI/CD). With the advent of containers, Kubernetes, and cloud architecture, they can quickly test, iterate, build, destroy and rebuild, all in a matter of minutes. If they have a question about building or fixing code, AI will instantly give them complete snippets of code, troubleshoot why code isn't working, or even give them real-time suggestions while they code.

Add to this that development organizations today can be huge, with thousands of developers, and grow exponentially almost overnight due to M&A activity — and you've got a dev environment that is more fast-moving and fluid than anything we've ever seen before.

While fostering an unprecedented level of innovation, this software development revolution has also greatly expanded application attack surfaces.

---

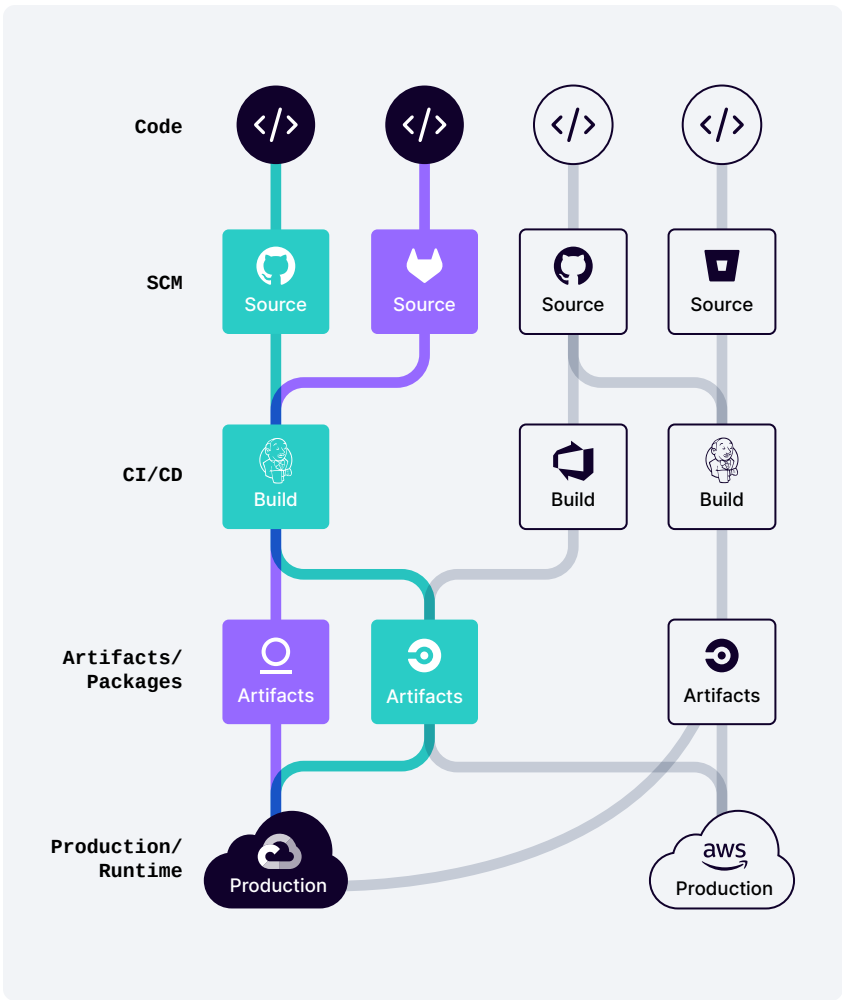
98%

**of enterprises  
deploy multi-cloud  
architectures, with  
data distributed  
across several  
cloud providers.**

---

72%

**of containers live  
less than 5 minutes,  
reported by Sysdig.**



# Security Challenges

## Introduced by Modern Software Development

.....

Modern software development  
is making the security team's job  
increasingly challenging.



Three emerging and especially vexing struggles:

1

Visibility

Lack of visibility into the full software factory, from assets to pathways and pipelines.



2

Correlation

Lack of correlation among types of risk — such as cloud, app, supply chain — across the SDLC, leading to increased manual efforts.



3

Complexity

Complexity leading to misconfigurations and the exposure of secrets in development pipelines.



## Lack of visibility and context make it hard to identify security risks and comply with regulations

With the explosion of innovations like APIs, multi-cloud deployments, containers, and AI-generated code, the attack surface has grown exponentially in recent years, and security teams are struggling to get the visibility and context they need to effectively mitigate risk.

---

**Generative AI gives developers an easier way to produce code at scale.**

**However, it also:**

- Creates a lower barrier of entry, allowing non-technical users to leverage tools such as GitHub Copilot to produce code.
- Generates code with vulnerabilities, just like code created by developers.
- Could include code licensed by another organization. This puts an enterprise at legal risk of stealing intellectual property.
- Often leverages old recursive patterns that developers have stopped using, but that are still on the Internet.

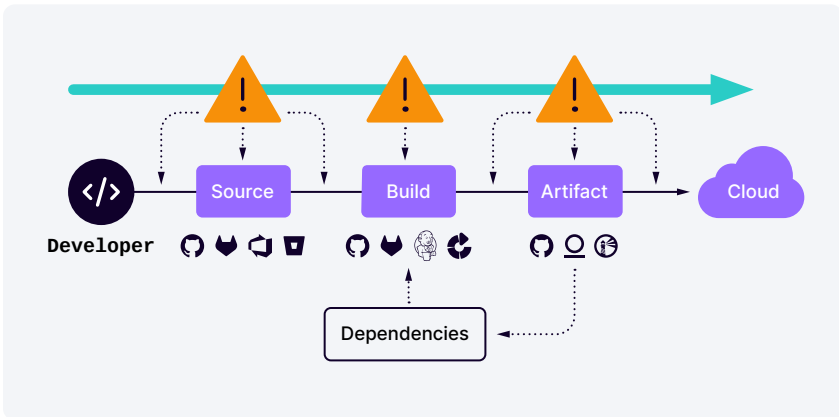
Beyond increasing risk, lack of visibility into the attack surface also creates compliance challenges. Security teams are struggling to comply with regulations requiring evidence of asset inventories and security controls.

---

**Lack of visibility makes complying with regulations more challenging.**

**For example, NIST SSDF requirements include the following:**

- 1 Prepare the Organization (PO)**
  - Implement roles and responsibilities in the SDLC.
  - Implement supporting toolchains.
  - Define and use criteria for software security checks.
- 2 Protect the Software (PS)**
  - Protect all forms of code from unauthorized access and tampering.
  - Archive and protect each software release.
- 3 Produce Well Secured Software (PW)**
  - Design software to meet security requirements and mitigate security risks.
  - Reuse existing, well-secured software when feasible instead of duplicating functionality.
- 4 Respond to Vulnerabilities (RV)**
  - Identify and confirm vulnerabilities on an ongoing basis.
  - Assess, prioritize, and remediate vulnerabilities.
  - Analyze vulnerabilities to identify their root causes.



### Correlation

## Silos prevent holistic view of product security

**Further complicating risk management are the silos among application, cloud, and security teams.**

Although most applications are deployed in the cloud, application security and cloud security are often two separate entities, leading to a lack of context and clarity. For instance, if a vulnerability is identified in a cloud environment, it can take hours for the cloud security team to work with the application security and development teams to locate the code creating the vulnerability.

Ultimately, neither of these teams are focused on the big picture of the software factory, leading to unidentified supply chain risk.

Developers are often also working in their own silo, spinning up new repositories or servers without the security team's awareness or visibility.

## Increasing complexity opens up new opportunities for misconfigurations and secrets exposure

**The complexity of the modern software factory also opens up new avenues for:**

1

Risky misconfigurations, such as of build systems

2

Exposure of secrets, such as API keys and cloud credentials

### Misconfiguration mayhem

With the software development factory becoming more complex and automated, there are increasing opportunities for misconfigurations to create risk. For example, misconfigured build servers is a common problem 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.

Oftentimes, development tools are over-privileged because they're easier to integrate if users have full access. In some instances, organizations spin up an open-source development server and then allow admin access to everything. They're not worried about misconfigurations; they're focused on application vulnerabilities.

In fact, this type of misconfiguration was the source of the SolarWinds and the Codecov attacks.

## Secrets in the spotlight

Modern apps require hundreds of secrets to function (API keys, third parties, 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.

---

# #2

**According to IBM's 2023 Data Breach Report, secret leak risks are the second most common initial attack vector.**

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.

---

# 12:100

**Legit Security has found an average of 12 secrets submitted per 100 repositories every week.**

# How Attackers Are **Taking Advantage**

.....

To take advantage of the vulnerabilities created by modern software development environments, sophisticated attackers have expanded their focus beyond front-end applications.

**Now, attackers also increasingly target software supply chain factory components (pipelines, build servers, libraries, tools, and processes).**

Expert nation-state attackers and professional cybercriminals know that infiltrating software supply chains and injecting malicious code is a one-to-many type attack that gets them more bang for their buck. The target organization will unknowingly send the code to hundreds or even thousands of customers or internal users downstream. Attacks like these have led to massive global breaches, such as those at 3CX, SolarWinds, Codecov, and CyberLink.

And those breaches have led to downstream customers being compromised as well, as was seen in the SolarWinds, Codecov, and recent Microsoft attacks. This could, and has in the past, allowed attackers to nest supply chain attacks, creating wide blast radiuses.

3CX was one of the first nested supply chain attacks. A developer from another company at Tradewinds (trading software) was compromised. The attacker injected malicious code into the software that was then downloaded by a developer at 3CX and allowed the attacker to breach the 3CX environment. They then moved laterally through the software supply chain and made malicious changes to code, which was then sent to 3CX clients, including Fortune 500 companies and the federal government.

---

633%

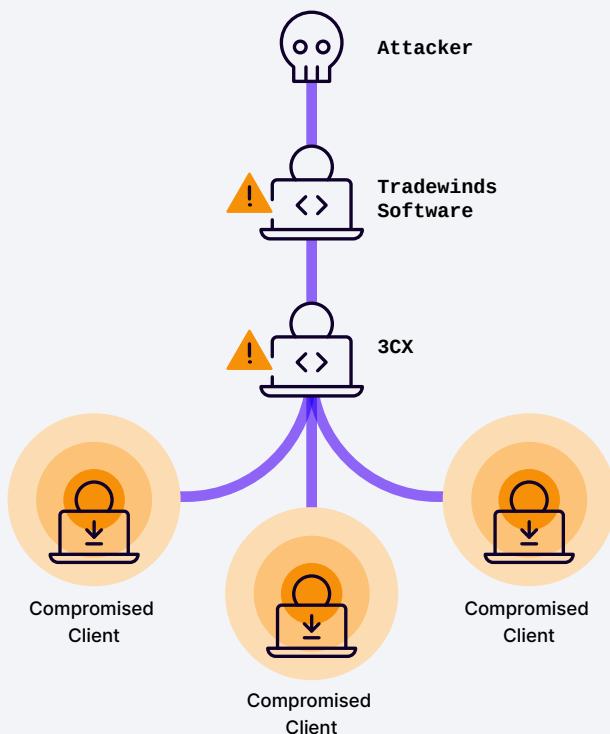
**increase in software supply chain attacks (88,000 attacks in a 12-month period), reported by Sonatype.**

---

61%

**of U.S. businesses were directly impacted by a software supply chain attack in the 12-month period ending in April 2023.**





**The impact of a software supply chain attack:**

- Lateral movement
- Intellectual property leak
- Customers' data leak
- Backdoors

There's still a need for strong application security controls inside of applications. However, the more devastating attacks are largely coming through software supply chains.

# Shortcomings of the Current Application Security Approach

.....

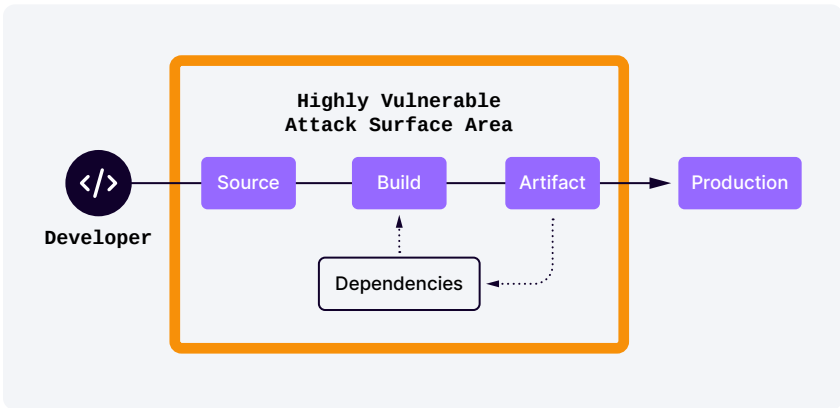
**Bottom line: The current approach to application security doesn't align with the way development works, or the way attackers do.**

Most security teams today approach application security by scanning source code in several ways:

- Static analysis (SAST)
- Dynamic analysis (DAST)
- Software composition analysis (SCA)
- Pen testing

These scans are conducted in different phases of code development and often by different teams — for instance, static analysis and SCA are often conducted earlier in the coding phase, whereas dynamic analysis, pen testing, and now next-gen SCA are typically seen later in runtime testing.

**This approach falls short because its focus is too narrow, it lacks context, and it produces a variety of results without correlation.**



## Correlation conundrum

It's time-consuming to correlate all the results coming from different AppSec tools. There are too many dashboards creating noise, and sometimes even conflicting or duplicate information. For example, with each producing its own list of weaknesses or vulnerabilities, SAST, DAST, and pen testing often produce overlapping results.



### Bottom line:

**You could end up chasing down the same issue in multiple tools.**

## Context crisis

With different tools scanning code in different ways across the development lifecycle, AppSec tools also generate a lot of results without a lot of context.

Security teams often assign developers all critical vulnerabilities to fix, but assigning developers to vulnerabilities just to lower a metric does not provide value if fixing those vulnerabilities doesn't impact the business or reduce the risk to the environment.

It's not a matter of just prioritizing which vulnerabilities to fix first based on the CVSS score of the vulnerability. It's also about assessing the business value of fixing vulnerabilities that are more risky because they impact high-value business applications that involve revenue, customer interactions, or sensitive data.

By relying only on tools to build an application security program, it's difficult to efficiently identify all the risks and understand how to remediate those risks in a way that is contextual to the environment. Without contextual risk ranking among applications and vulnerabilities, either "everything" is a priority, or "nothing" is a priority.


Additionally, time, money, and effort could be spent fixing something deemed critical by an outside entity that doesn't actually represent real risk to the organization, whereas other vulnerabilities or weaknesses that are risky to the organization are overlooked due to low classifications from things like CVSS or EPSS.

## Focus failure

Scanning source code is a critical part of application security, but it's only a part, and it needs to be streamlined.

"Shift left" is the right approach, but when you shift everything left, not just the critical issues, developers are inundated and don't have a clear focus.

In addition, most AppSec tools only look at application risk and largely ignore the risk found in the software factory, like weaknesses in the CI/CD pipeline. This focus leads to blind spots in the area that's currently causing and producing the most devastating attacks seen in the wild.



**Ultimately, application security tools work more effectively and efficiently with a solid foundation underneath them.**

# A New Approach to Security: **Visibility, Prioritization, and Alignment**

.....

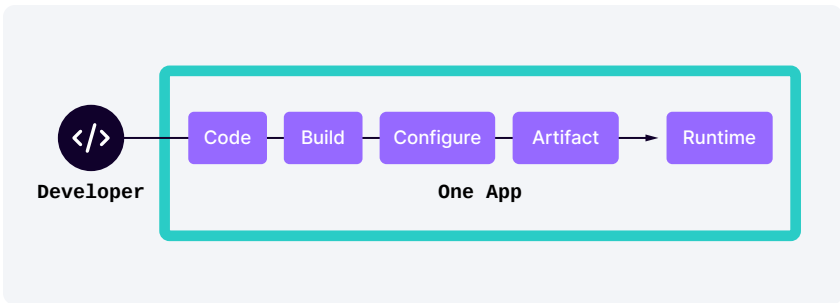
To sync with the way today's attackers  
and developers work, enterprises must  
break down their security silos.

Breaking down security silos requires focusing the mission on overall product security — rather than creating silos for cloud security, application security, and other components of the software supply chain.

It also calls for one platform that can corral the chaos, scale as development organizations grow and change, and offer a clear view of the full software factory, its assets, its owners, its security controls, its vulnerabilities, and how all are related.

**Benefits of Application Security**  
**Visibility, Prioritization, and Alignment**

Mitigates high-priority security vulnerabilities to intelligently reduce risk.	Streamlines regulation compliance by proving where controls are deployed.
Uncovers shadow IT, systems, and source code.	Evaluates application business criticality.
Measures the blast radius of vulnerabilities—the potential impact of a security breach within a system.	Provides a common language for executives, developers, and security teams to understand risks.
Provides guardrails that allow developers to move fast without security controls slowing them down.	Shows progress in reducing risks.



## One App

**Enterprises need an approach that would take application security and cloud security and look at them through the lens of one application.**

This shift includes rethinking the definition of an app to include stages from code, through the build process, configuration, artifact management and into runtime.

To protect against software supply chain attacks and comply with regulatory mandates, enterprises should consider a comprehensive application security program that simultaneously assesses and addresses all risks across the entire SDLC holistically. The risks found in one area of an SDLC interrelate with risks found in every other area of the SDLC due to the nature of the way cloud native applications are built.



# From Code to SDLC

**Rather than assuming development pipelines are secure, assume they are insecure until proven otherwise.**

The first goal should be to determine which existing risks require the most immediate attention. Identify all the components and harden the entire SDLC, not just the code.

Establishing this visibility is similar to the current approach to endpoint and network protection — where security teams collect a complete list of assets and a map of the environment to know what to protect.

Creating an automated SDLC asset inventory for development pipelines will allow security teams to apply security policies to that inventory and check for risky violations — quickly and accurately. Threat modeling will then allow the team to identify risky implementations before starting development.

To enable comprehensive application security, aim for:

---

## Centralized Visibility

*across all phases of the software factory*

---

**This gives everyone an end-to-end understanding of all SDLC assets and allows them to answer the key questions about applications, products, build tools, and controls:**

- Who's developing the code?
- What's in your code?  
(i.e., secrets, AI/LLM, vulnerabilities, sensitive data)
- Do we know when new build assets or artifacts are created (preventing shadow IT)?
- Do we have a complete SDLC asset inventory?
- Can we verify correct configurations across all tools throughout the code factory?
- Where and how does code go through pipelines?
- Where does code end up in production environments?
- How is code protected in pipelines and production environments?
- Can we triage and prioritize issues based on business risk?

**As these questions are answered, security teams can see where security gaps exist and more easily comply with regulations.**

---

# Contextual Prioritization

*of the overall development environment*

---

**This allows teams to make the right security decisions by:**

- Determining the risks in each stage of the SDLC.
- Identifying how those risks correlate across the software factory.
- Selecting which vulnerabilities to fix first based on the impact on the business.

Avoid assuming all critical vulnerabilities must be prioritized. They may be associated with low-level applications and code repositories.

---

# Organizational Alignment

---

**To sync the knowledge and the efforts of developers, security teams, and executives to:**

- Explain where the risks are and why.
- Identify the required resources to fix risks.
- Inform why it's important to fix certain vulnerabilities and put others on the back burner.
- Strengthen internal relationships by providing dashboards showing vulnerability status.

Everyone realizes which risks need to be fixed and why. This is more powerful than using arbitrary numbers from an outside entity that doesn't have any idea how the business is run.

# Don't Install Security Cameras Before You **Know the Layout of the House**

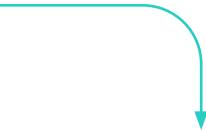
.....

Scanning code without a solid foundation of SDLC visibility is like installing security cameras without first understanding entrances, exits, high-traffic areas, or where valuables are kept.

Sophisticated cybercriminals now focus their efforts on the expanding attack surfaces of software supply chains. Make sure they don't understand the "layout of the house" better than you do. It's time for security strategies that go beyond scanning of applications to proactively lay a foundation of centralized risk visibility and vulnerability prioritization.

The best way to secure applications from coding to the cloud is real-time application security posture management that provides visibility, prioritization, and alignment among internal teams to ensure the integrity, governance, and compliance of every software release.

It's not enough to just manage security issues; enterprises must prevent issues by centralizing security policy enforcement across all applications, teams, and pipelines — with real-time updates on security issues, configuration changes, and compliance drift.



**Learn how enterprises like Kraft-Heinz and Netskope are taking a new approach to application security.**

Visit our [website](#) to book a [demo](#)  
and learn more about the Legit  
Security Platform.

Get best practices on software  
supply chain security from our [blog](#).  
Follow us on [LinkedIn](#) for the latest  
news, events, and content.

