



EBOOK

4 STEPS TO SECURING YOUR SOFTWARE SUPPLY CHAIN

ADDRESSING SECURITY VULNERABILITIES IN DEVELOPMENT ENVIRONMENTS

THE VULNERABLE SOFTWARE SUPPLY CHAIN

Advanced, software supply chain attacks have a vast and rippling impact. By injecting malicious code into an otherwise legitimate software update, bad actors infected over 18,000 conscientious SolarWinds customers.

The malware inserted in SolarWinds' Orion application is just one vector of what looks to have been a well-planned, multi-pronged campaign targeting specific organizations.

Such a high impact breach exposes the increasing attack surface and vulnerability of software development and delivery. With the advent of CI/CD pipelines, supply chain attacks have become more prevalent – with attackers compromising certificates to sign code and bypass controls.

- As early as 2016, the BitTorrent client Transmission's source code was backdoored on GitHub. And in 2017 the popular cleanup application Ccleaner was backdoored via a compromised code signing certificate
- A [Docker Hub breach](#) allowed the theft of 190,000 usernames and hashed passwords and exposed Bitbucket and Github access tokens
- A [Kubernetes security flaw](#) allowed attackers to use an infected container to replace files on users' workstations.

Contents

The Vulnerable Software Supply Chain	2
Increased Risk in the DevOps Landscape	5
Step 1: Secure Developer Workstations.....	7
Step 2: Protect Application Credentials.....	9
Step 3: Safeguard DevOps Tool Administration Consoles.....	11
Step 4: Weave the Three Strategies to Work Together	13
Securing the Development Environment is a Team Sport	15
Next Steps – How CyberArk Can Help.....	17



Layered Identity Security Controls

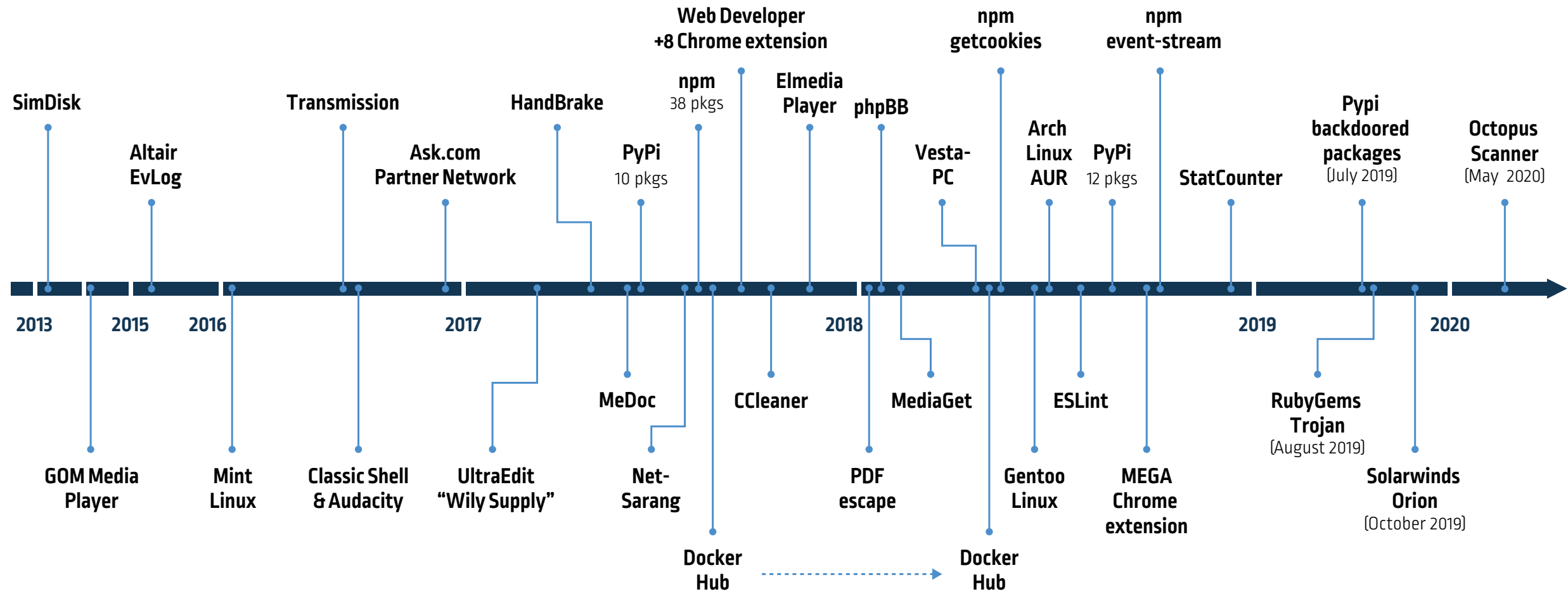
Enterprise software development and delivery has become highly automated with minimal human intervention during the build process. Many development tools and platforms themselves are Tier 0 assets, with credentials and access to other assets across the enterprise and from third parties.

The development environment has become more interconnected, frequently with cloud-based components outside the direct control of the developers, operations, or security.

Still, the majority of all cyberattacks involve – and ultimately rely upon – the compromise of identity and the manipulation of privileged access. The SolarWinds breach is no exception.

This eBook explains how best practices for layered identity security controls in the development and delivery environment go a long way in limiting both, vulnerabilities in the software supply chain – and attackers from accessing, stealing, and controlling enterprise assets.

A Timeline of Increasing Supply Chain Attacks



INCREASED RISK IN THE DEVOPS LANDSCAPE

Talented developers, both internal and external, are key to successful digital transformation initiatives. With more automated software development and delivery processes and access to cloud-based tools and repositories, they can deliver more code faster.

But these advances also expand the attack surface. There are more vulnerabilities along the software supply chain, from coding with open-source components, to automated builds and testing, to cloud-based deployment. This is especially evident with insecure privileged accounts, credentials, and secrets, e.g., API calls, encryption keys, access tokens, certificates, passwords, etc.

Software supply chain attacks, like the SolarWinds campaign, can be multi-layered – involving several interim steps or vectors. But they have in common the ultimate goal: to compromise identities and escalate privileged access.

A fundamental component of a defense-in-depth (DiD) approach is securing identities throughout the development environment, including cloud native applications. We will next cover the four most important strategies in a DiD approach to securing CI/CD.

4 Steps to Securing Your Supply Chain

1

Securing development workstations and endpoints

2

Protecting application credentials

3

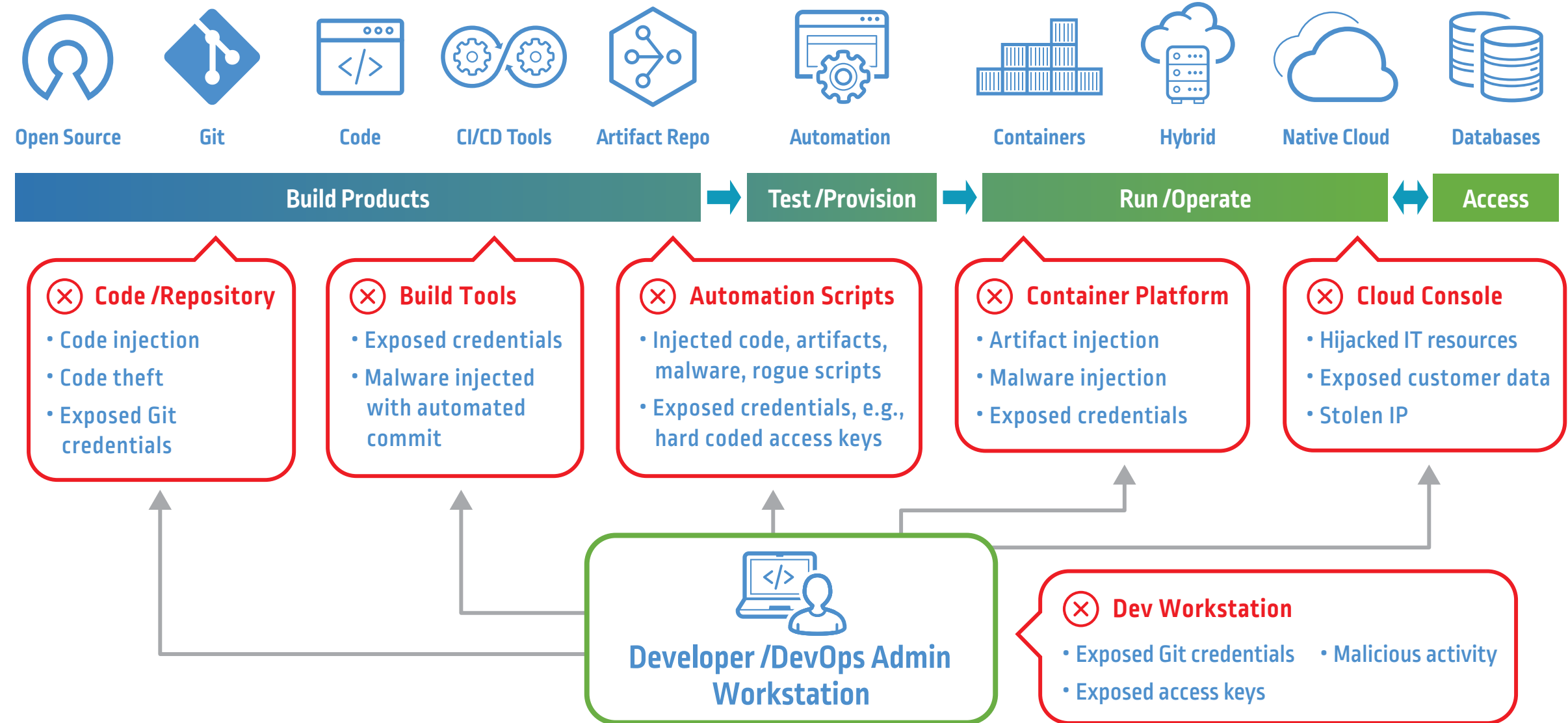
Safeguarding DevOps tool administrative consoles

4

Finally, combining these best practices into a holistic approach to securing development environments

And as we shall see, the evolving development landscape and resultant threats to the enterprise make it imperative that Development and Security leadership work together.

Vulnerabilities in the Software Supply Chain





Step 1:

SECURE DEVELOPER WORKSTATIONS

Developers need access to be productive and not be burdened by security. The #1 goal is to make security transparent; to allow access to assets and services they require, yet smartly restrict access to what they do not. Developer workstations can be tiered according to usage and corresponding security requirements.




Whether accessed locally from a desktop or remotely via laptop, some DevOps processes and systems require a high level of privilege, if only for some specific actions and for limited time periods.

One of the biggest dangers is these credentials are often saved locally, making developers' workstations high-value targets for attacks from something as simple as a phishing email.

The most effective strategy for protecting privileges on developer workstations follows these best practices:

1. Remove local admin rights and apply the principles of Zero Trust and Least Privilege to ensure privileges are elevated only when needed.
2. Use MFA (Multi-Factor Authentication) when possible.
3. Use install privileges to block installation of unknown apps and allow approved tools. Remove unapproved software and flag violators.
4. Secure local credential stores, i.e., protect default locations where cloud access keys, such as AWS Access Keys and Git credentials, are stored on endpoints.
5. Block known malware and prevent potential compromise by running unknown apps in restricted mode.
6. Review restrictions on a regular basis with the goal of minimizing privilege while avoiding overly onerous restrictions which inhibit developer productivity.

Secure Developer Workstations – Tiering Example

<p>TIER 1: Top tier for power users</p>	<p>Highest requirements</p> <ul style="list-style-type: none"> • On demand privilege elevation • Self-service with management approval • Only permit installs that meet policy • Flag violators 	<p>10% of Workstations</p> 
<p>TIER 2: Middle tier-most users</p>	<p>Some special requirements</p> <ul style="list-style-type: none"> • Edit environmental variables • Edit host files • Use Visual Studio 	<p>50% of Workstations</p> 
<p>TIER 3: Lowest tier</p>	<p>Basic user with few special requirements</p>	<p>40% of Workstations</p> 

Step 2:

PROTECT APPLICATION CREDENTIALS

Virtually all applications use credentials to access other assets and resources throughout the enterprise. The SolarWinds episode has made abundantly clear why the enterprise needs to secure all their applications everywhere, across the entire organization.

Weakly or inconsistently stored credentials in applications, scripts, and other sources is a major vulnerability. Frequently these credentials cannot be easily rotated, monitored, or tracked. They can also be exposed inadvertently if the code is posted to a public site such as GitHub.

Establish the following best practices for securing the credentials used by applications, scripts, and other non-human identities to securely access databases and other sensitive resources:

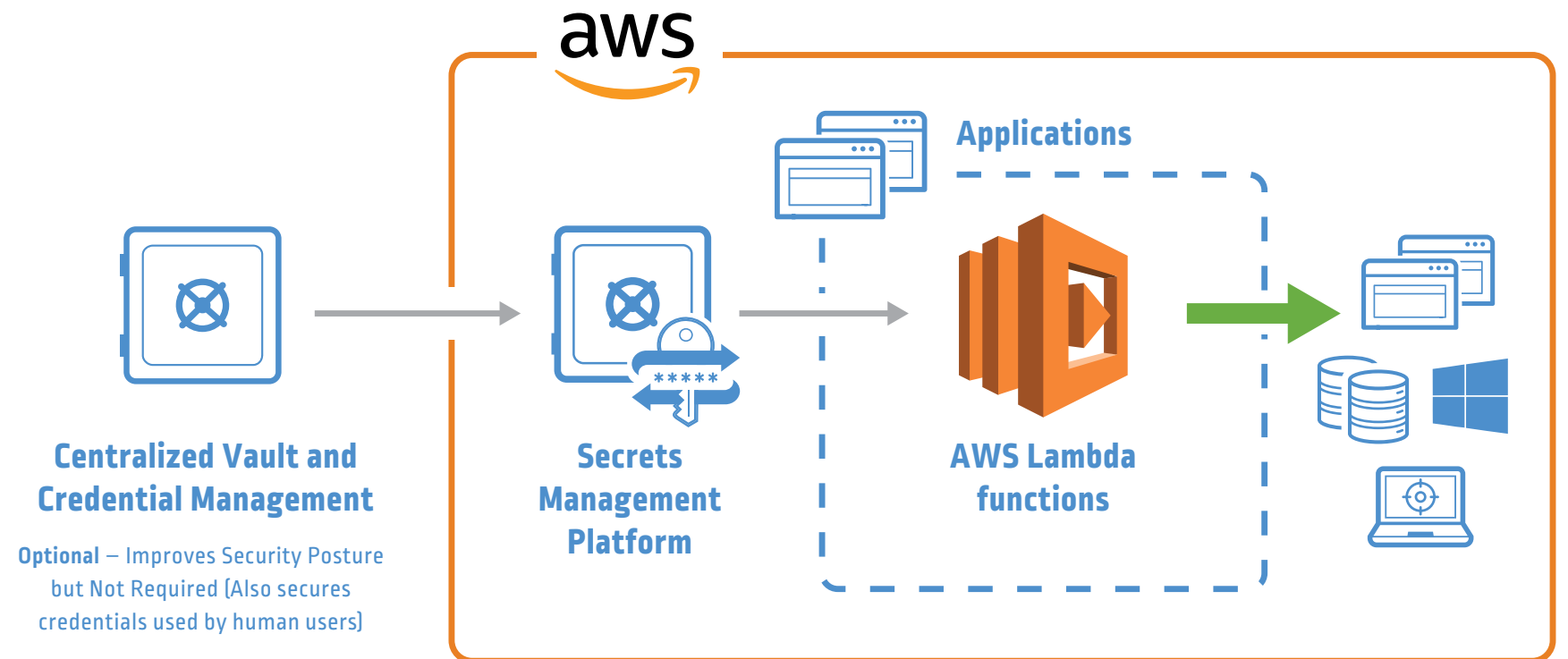
1. Eliminate all hard-coded credentials used to access databases, tools, and other sensitive resources. Replace with a secure approach, such as an API call to fetch the secret from a digital vault.



2. Centrally rotate, manage, store, and monitor secrets and other credentials used by applications, scripts, and other non-human identities. Ensure audit and compliance needs can be met.
3. Strongly authenticate applications, containers, and other non-human identities requesting access to secrets and credentials. Whenever possible use the native attributes of the requestor for authentication to eliminate the secret zero issue.
4. Apply principles of least privilege and role-based access controls (RBAC) so the application, container, script, or automation process only has access to the credentials it needs.

These best practices are far easier to implement, and manage going forward, with a centralized, secrets management platform.

Example of Securing Application Credentials





Step 3:

SAFEGUARD DEVOPS TOOL ADMINISTRATION CONSOLES

Widely used DevOps tools and platforms such as Jenkins and Azure DevOps; provisioning tools like Ansible and Puppet; and container-orchestration environments like Red Hat OpenShift, Kubernetes, and VMware Tanzu provide extraordinary control over an organization's development resources. They also can provide far-reaching access to enterprise resources beyond development.

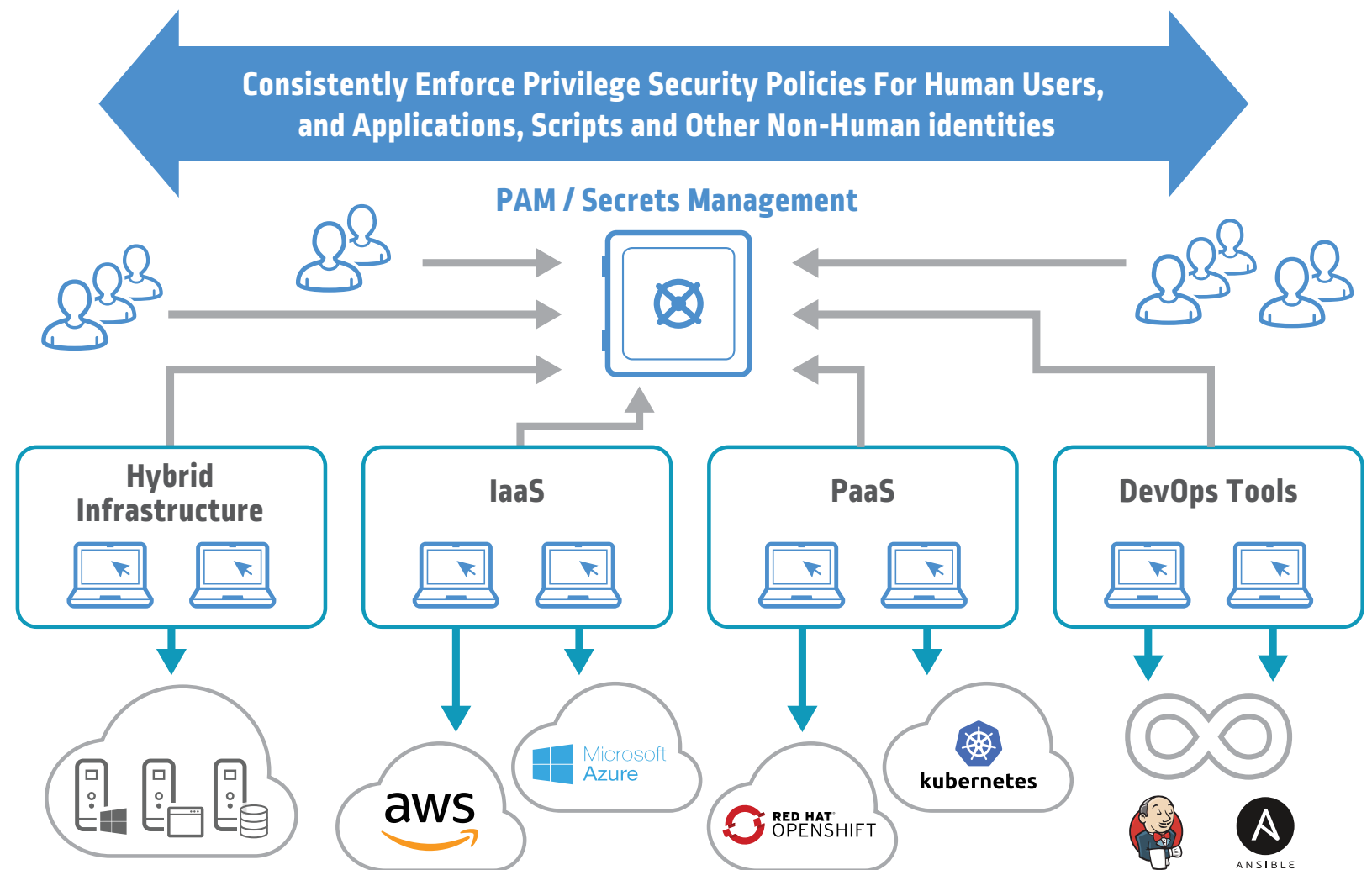
As the usage of these administrative consoles surges, they have become attractive targets for attackers, who can exploit unsecured consoles to target other resources in the development environment and beyond. Far too many organizations use the default configurations of these tools, which in some cases do not require passwords for privileged access. Attackers are using bot crawlers to systematically search out and exploit these exposed vulnerabilities.

While vendors and open-source communities have worked to address these vulnerabilities, security weaknesses remain.

Secure DevOps tool administrative consoles with these best practices:

1. Apply basic password best practices (manage, rotate, etc.), and use MFA for critical operations. Ideally use MFA for all console access. Avoid the use of default configs and passwords. Example: some tools establish a developer default user to create projects, other tool consoles can be accessed using http or with the default password.
2. Centrally control and manage human and other interactive access to management consoles and Command Line Interfaces (CLI). Example: securing access to the Jenkins UI (User Interface).
3. Transparently manage and monitor sessions and make it easy for developers to adopt secure solutions. Limit the attack surface by providing access to the CLI only via jump servers and other monitoring tools.
4. Use adaptive, context-based MFA to maintain productivity yet mitigate risk. Step up authentication forces human review or approval before sensitive scripts are run automatically, such as pushing commits to Git.

Secure Developer Workstations – Tiering Example



Step 4:

WEAVE THE THREE STRATEGIES TO WORK TOGETHER

A holistic approach combines these best practices to secure the full development environment, and build a layered, DiD for the enterprise. This calls for Development and Security leadership to work together to establish policies – and enforce adoption.

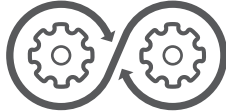






The SolarWinds episode is an opportunity to address vulnerabilities in the development environment that expose the entire organization. Developers and security teams are motivated to ensure no malicious code or artifacts can be injected into their code, as well as protect their intellectual property.



The development environment of each organization is different, but typical successful strategies that employ the best practices described above include:

1. Segment the code by development team and use MFA to force intervention and review of sensitive automated actions, e.g., commits to master.
2. Development and security team together establish a hierarchy of end user requirements. For example who needs on-demand privilege escalation or self-service with management approval.
3. Establish best practices for the securing of credentials used by applications, scripts, and other non-human identities. This includes eliminating hard-coded credentials; centrally store, rotate and manage; and to strongly authenticate applications, containers, and other non-human identities requesting access to secrets and credentials.
4. The security team regularly and proactively adjusts endpoint protections to accommodate developer needs. It is a matter of balancing protection and productivity.

Secure the Full Development Environment

<p>Control Point 3: Apply Policies</p>	<p>Enforce polices based on best practices</p> <ul style="list-style-type: none"> • Enforce consistent policies • Segment access to code base • Trigger manual review after pull request • Discard build environments after single use 	
<p>Control Point 2: Secure Credentials & Access</p>	<p>Control access to dev tool consoles</p> <ul style="list-style-type: none"> • Secure tools and scripts • Provide positive dev experience • Monitor pull requests  	<p>Manage application secrets and credentials</p> <ul style="list-style-type: none"> • Secure secrets in code and automation scripts • Rotate, manage and audit  
<p>Control Point 1: Secure Endpoints</p>	<p>Secure developer workstations</p> <ul style="list-style-type: none"> • Apply least privilege and privilege escalation • Block storage of Git and Jenkins credentials • Enable teams to select preferred IDE and dev tools  	

SECURING THE DEVELOPMENT ENVIRONMENT IS A TEAM SPORT

Let's be honest: any security strategy that slows down the delivery of code encourages developers to get creative with workarounds. It is also true that any security strategy for development cannot be so burdensome that it swamps security teams with difficult to manage processes for workstations, applications, and increasing DevOps tools (many including automation).

Development environments are complex and securing them requires a specific, focused perspective. Developers may not fully recognize all the potential vulnerabilities that could impact the organization. By the same token, IT and security teams may be less familiar with development tools and processes.

It is critical that security teams and development work together. And it is essential that Development and Security leadership make it a priority to enable this cooperation.

Approaches to Engaging Developers

Security team members know that they need to work more proactively with their Dev and DevOps counterparts but often don't know where to start. Research on approaches for securing DevOps environments from CISOs and security leaders at the Global 2000 identified five key approaches to successfully engage with and help developers adopt secure practices:

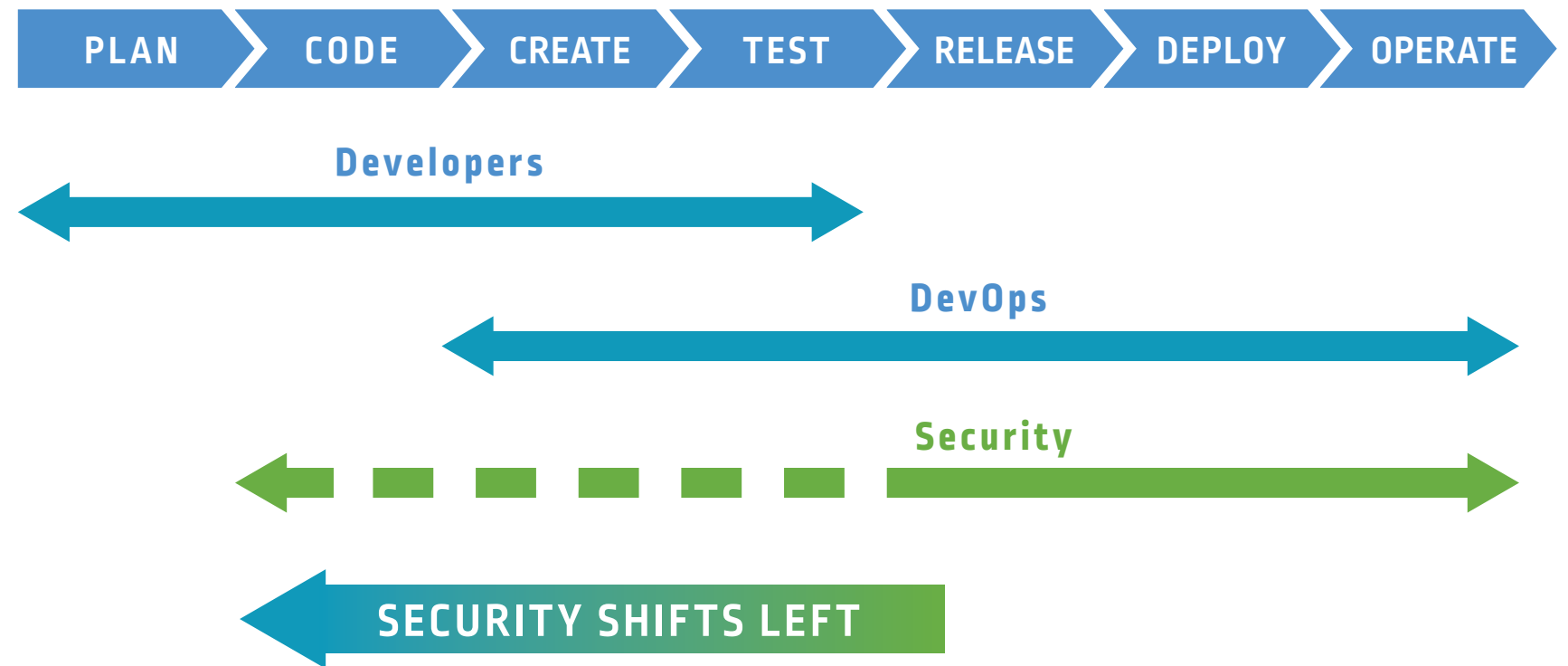
1. Transform the security team into a DevOps partner.
2. Prioritize securing DevOps tools and infrastructure.
3. Establish enterprise requirements for securing credentials and secrets in DevOps and cloud environments.
4. Adapt secure processes for application testing.
5. Evaluate the results of the DevOps security program.

[LEARN MORE](#)

Security teams can more effectively work with developers to secure applications by proactively reaching out earlier in the development process, commonly referred to as “shifting left.” Conversely, developers can make it a practice to call in security at the planning stage—rather than waiting to get them involved just before the code goes live. Shifting security left to more fully include the software supply chain is critical to protecting the enterprise.

Protecting privileged access embodied in credentials and secrets, on developer workstations, within applications, and on admin consoles, is key to securing the software supply chain. Securing identity throughout the development environment, increasingly automated and a doorway to other assets, is fundamental to an enterprise DiD strategy.

Shifting Security Left



NEXT STEP – HOW CYBERARK CAN HELP

CyberArk solutions enable you to take a proactive, holistic approach to securing privilege throughout your development environment.

No other vendor has the broad experience or offers a comparable breadth of privileged access management solutions for securing both human users and non-human identities across the entire organization – including DevOps, hybrid, and native cloud environments.

- [CyberArk Secrets Manager](#) solutions integrate with popular tools and container platforms to provide a comprehensive solution for securing secrets and other credentials in DevOps environments.
- [CyberArk Endpoint Privilege Manager](#) enforces least privilege on the endpoint (Window and Mac) early in their lifecycle. It enables revocation of local administrator rights and implements application controls while minimizing impact on user productivity.
- [CyberArk Privileged Access Manager](#) enables organizations to secure the privileged credentials and secrets used across their entire enterprise, including the development environments that are driving the organizations digital transformation.
- [CyberArk Adaptive Multi-Factor Authentication](#) facilitates implementation of a broad range of secondary authentication methods across your entire organization.

CyberArk Blueprint

CyberArk offers a prescriptive framework to help guide organizations with a risk-based approach to building their privileged access management programs.

Blueprint not only address enterprise-wide on premises systems but also infrastructure-as-a-service admins, CI/CD consoles, and dynamic applications. This ensures privilege is secure wherever it exists across the application development and deployment process.

[LEARN MORE](#)

THIS PUBLICATION IS FOR INFORMATIONAL PURPOSES ONLY AND IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER WHETHER EXPRESSED OR IMPLIED, INCLUDING WARRANTY OF MERCHANTABILITY, FITNESS FOR ANY PARTICULAR PURPOSE, NON-INFRINGEMENT OR OTHERWISE. IN NO EVENT SHALL CYBERARK BE LIABLE FOR ANY DAMAGES WHATSOEVER, AND IN PARTICULAR CYBERARK SHALL NOT BE LIABLE FOR DIRECT, SPECIAL, INDIRECT, CONSEQUENTIAL, OR INCIDENTAL DAMAGES, OR DAMAGES FOR LOST PROFITS, LOSS OF REVENUE OR LOSS OF USE, COST OF REPLACEMENT GOODS, LOSS OR DAMAGE TO DATA ARISING FROM USE OF OR IN RELIANCE ON THIS PUBLICATION, EVEN IF CYBERARK HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

03.21. Doc. 212312