Symantec
A Division of Broadcom

White Paper

Secrets Management: Challenges and Best Practices

by Joseph Burke, Chief Architect, Privileged Access Management

Introduction

In the technology world, the simplest definition of a secret is something you want to protect from public knowledge while being able to use it to authenticate one thing to another and/or to identify and trust something foreign.

Some examples include:

- A user's credential when logging into a server or a website
- Database identity and password when an application connects
- Remote service when called by a script or an application
- IoT device connecting to a centralized resource
- Security token
- CA certificate to secure a resource

Secrets take on many forms, from a simple user name and password, to an SSH key, to a short-lived token like RSA or OAuth tokens, to a private encryption key, and so on. Secrets not only provide the authenticated identity but can also provide a set of specific authorizations for the connection. Secrets are used everywhere: by employees, vendors, partners, customers, automated processes (DevOps), third-party vendors (integrations), servers, local services, applications, scripts, APIs, Cloud providers, SaaS services, and so on to obtain privileged access to perform an elevated or protected function.

While many different types of secrets exist and continue to emerge, fundamentally, a secret in the wrong hands leads to a significant security event and/or breach. In the news, many of the recent public breaches over the past decade are due to a mishandling of a secret (like Yahoo, Marriott, Wawa, and so on).

Information Security specialists at every company are responsible for deriving and maintaining the rules on how secrets are used across the enterprise. Given the broad scope of landscape to cover, many take an initial approach to control privileged access using vaulted secrets to protect the most critical functions within their business, operations, technology, and infrastructure. Once vaulted, control then expands to include a series of rotation policies based on an expiry date and/or key access events.

Security teams are challenged to ensure an adequate management of secrets throughout the organization. This includes:

- Understanding where all secrets exist: Building a centralized catalogue
- Defining policy on how to manage secrets: Frequency of rotation, determination of strength, and emergency access
- Understanding the tool sets that exist to help mitigate these risks and automating as much as possible
- Rolling out a program to protect the secrets throughout the organization
- Enforcing control throughout the organization to minimize risk
- Requiring managers to attest to the following:
  - Who has the potential access to secrets
  - Who has used a secret and when

Unfortunately, common and convenient practices do not often consider security until very late in their development or implementation cycle, leaving the company, product, and intellectual property potentially vulnerable.

The following are some best practices and capabilities to consider when looking for tools to manage secrets:

- Discover an inventory of assets and secrets throughout your organization: In Microsoft Active Directory or LDAP, local credentials, service credentials, and so on.
- Securely catalogue and vault each secret into a central encrypted, hardened location.
- Practice a zero-trust security posture by trusting nothing. Access to the secret should be governed by a least privileged, request-contextualized and risk-aware process.
- Change or rotate the secret often—on demand or after each use.
- Protect the use of secrets from end users as much as possible by providing auto-login capabilities; however, if secrets are provided to users then ensure policies automatically rotate immediately after use.
- Protect the secret from applications and configs at all times—obtain the secret at runtime (requested on-demand when needed) versus adding the secret at build, config, or deployment time where the secret can be accidentally observed while accessing the underlying infrastructure.

Methodologies for Managing Secrets

A variety of methodologies exist on how to manage secrets, some are more practical than others:

- Managing manually
- Leveraging platform-specific tools
- Using a single, centralized source across all platforms
- Operationally accessing the secret when needed at runtime versus build or deploy time

Manual Management

This method does not scale and is prone to human error; this should be avoided.

Platform-Specific Tools

Tooling exists for platform-specific solutions like in Kuberentes, Cloud providers, Docker: Niche vendors that perform a single function. Managing multiple locations and toolsets can result in a heavy operational burden on security teams plus may lead to policy inconsistencies when it comes to the strengths and capabilities of each tool.

NOTE: Check what methodology each tool is using and assess how easy or difficult it is to compromise the secret from the underlying infrastructure.

Single, Centralized Source Across all Platforms

PAM solutions provide a single, centralized source to vault and provide zero-trust access to secrets. A single solution is preferred to ensure consistency—a single-point control for access and a single source for auditing who has accessed the secret.

Access at Runtime vs. Build or Deploy Time

Many secret management solutions provide the secret at build or deploy time and not runtime. There are some use cases where this is safe, like providing a CA Certificate, because the secret is not rotated very often and is likely widely used across many devices.

However, this presents a weakened security posture by delegating the secret's security to the infrastructure or the DevOps pipeline and infrastructure that the company may not be directly operating.

For example, if a container is running in a Kubernetes cluster and leveraged a DevOps tool to provide the secret during the deployment, in all likelihood, the secret was added as an environment variable in the container.

This successfully solves one problem of the secret not being saved inside a source control system (SCM). However, anyone with access to the underlying infrastructure and/or Kuberentes nodes may have unauditable access to view the credential by typing a *docker inspect* on the container, which shows the environment variables and exposes the secret.

To solve this, the underlying Kubernetes nodes must be managed in the PAM solution to provide zero-trust access. This becomes even more challenging as the world shifts to serverless and cloud-provided kubernetes services (for example, AKS) where the infrastructure is not known nor managed by the organization. How can a company's security team track, manage, and audit this?

A more secure way is to provide runtime secret access—a programmatic way to request the secret directly from the code (script, application, service, and so on) where trust is established and known to a PAM solution. The secret is protected and provided on-demand at runtime—not at config, build, or deploy-time.

Conclusion

Security teams are chartered to provide control over how secrets are used throughout an organization while trying to keep up with the rapidly changing technical landscape.

Implementing a zero trust approach while vaulting secrets in a hardened vault, providing a full audit trail of potential and actual secret use, and controlling when secrets are changed are some of the key best practices that mature PAM solutions provide.

The best PAM solutions go further to ensure the correct controls are in place for both human-based and application-based secret use. Having the ability to protect, authorize, and plug in to the infrastructure are essential concepts for security teams while simultaneously covering and transforming DevOps into DevSecOps workflows.