Understanding and Selecting a Secrets Management Platform

Version 1.0

Released: January, 2018

Securosis, L.L.C. 515 E. Carefree Highway Suite #766 Phoenix, AZ 85085 **T** 602-412-3051

info@securosis.com www.securosis.com

Author's Note

The content in this report was developed independently of any sponsors. It is based on material originally posted on the Securosis blog, but has been enhanced, reviewed, and professionally edited.

Special thanks to Chris Pepper for editing and content support.

Licensed by CyberArk

CyberArk (NASDAQ: CYBR) is the global leader in privileged account security, a critical layer of IT security to protect data, infrastructure and assets across the enterprise, in the cloud and throughout the DevOps pipeline. CyberArk delivers the industry's most complete solution to reduce risk created by privileged credentials and secrets. The company is trusted by the world's leading organizations, including more than 50 percent of the Fortune 100, to protect against external attackers and malicious insiders. A global company, CyberArk is headquartered in Petach Tikva, Israel, with U.S. headquarters located in Newton, Mass. The company also has offices throughout the Americas, EMEA, Asia Pacific and Japan. To learn more about CyberArk, visit www.cyberark.com, read the CyberArk blogs or follow on Twitter via @CyberArk, LinkedIn or Facebook.

Copyright

Understanding and Selecting a Secrets Management Platform

Table of Contents

Introduction

Secrets Management platforms do exactly what the name implies; they store, manage, and provide secrets. The technology addresses several problems most security folks don't yet know they have. As development teams leverage automation and orchestration techniques they create new security issues which then must be tackled. Our reliance on automation of cloud services is making application and IT services faster and more resilient, but with that advancement comes a need to provide permissions to machines and software instead of just people. In some ways this is the new iteration of the identity management problem from the early 2000s, magnified across personal devices and cloud computing infrastructure.

And that is the genesis of the problem: Developers have automated software build and testing, and IT automates provisioning, but both camps still believe security slows them down. Continuous Integration, Continuous Deployment, and DevOps practices all improve agility, but also introduce security risks — including storing secrets in source code repositories and leaving credentials sitting around. This bad habit leaves every piece of software that goes into production is at risk!

All software needs credentials to access other resources; to communicate with databases, to obtain encryption keys, and access other services. But these access privileges must be carefully protected lest they be abused by attackers! The problem is the intersection of knowing what rights to provision, what format the software can accept, and then securely provision access rights when a human is not — or cannot — be directly involved. Developers do integrate with *sources* for identity — such as directory services — but are usually unaware technologies exist that helps them distribute credentials to their intended *destinations*.

To address these changes, powerful utilities and platforms have been developed to secure sensitive data, and secure distribution of privileges. The term for this new class of product is "Secrets Management"; and it is changing how we deliver identity, secrets, and tokens — as well as the way we validate systems for automated establishment of trust. In fact Secrets Management is a core piece to move DevOps into SecDevOps. This research will explore why this is an issue for many organizations, what sorts of problems these new platforms tackle, and how they work in newer environments.

The Challenge of Machine Identities

Obtaining secrets is essential for automation scripts to function, but many organizations cling to the classic (simple) mode of operation: place secrets in files or embed them into scripts so tasks can complete without human intervention. Developers understand this is problematic and that this is a rapidly expanding form of technical and security debt that is being swept under the rug only to have it reappear in an audit or after a breach event. And they certainly ***do not*** go out of their way to tell security about how they provision secrets, so most CISOs and security architects are unaware of this emerging issue until an event occurs.

The problem is not new. Administrators have been putting secrets in unsecured files for decades. No administrator wants to be called into work in the middle of the night to enter a password so an application can restart. IT administrators routinely store encryption keys in files so an OS or application can access them when needed. Database administrators place encryption keys and passwords in files to facilitate automated reboots. Or they did until corporate networks came under even more attack and this industry-wide practice was disallowed. Since then we have relied upon everything from manual intervention, to key management servers, and even hardware dongles to provide a root of trust to establish identity and provision systems. But the old approaches don't provide the required security in new compute and development environments – and the stakes are much higher because of the dynamic nature of how we provide software and services.

It sounds cliche, sure, but IT and application environments are genuinely undergoing radical change. New ways of deploying applications as microservices and into containers are improving our ability to cost-effectively scale services and large systems. Software-defined IT stacks and granular control over services through APIs provide tremendous agility advantages. Modern operational models such as Continuous Integration and DevOps amplify these advantages, bringing applications and infrastructure to market faster and more reliably.

Perhaps the largest change currently affecting software development and IT is cloud computing.The on-demand and elastic nature of cloud-based services offers huge advantages, predicated on automated infrastructure defined as software. The cloud is not a necessary component of these other advances, but it makes them even more powerful. Leveraging all these advances together, a few lines of code can launch —or shut down — an entire (virtual) data center in minutes, with minimal human involvement or effort.

Alongside their benefits, automation and orchestration raise new security concerns. The major issue today is securely sharing secret information. This is especially problematic where 'machines' are not something sitting in a rack at a co-location facility, but instead an ephemeral instance of a machine — or perhaps ***thousands*** of instances running simultaneously, potentially being created and destroyed by the dozen due to the transitory vagaries of demand. How can we track which server, virtual machine, or container is which, or what set of permissions to provide each? And the scope of the problem broadens as we extend automation and orchestration across teams. Development teams need to share data, configurations, and access keys across and between teams to cooperate on application development and testing. Automated build servers need access to source code control, API gateways, and user roles to accomplish their tasks. Servers need access to encrypted disks, applications need to access databases, and containers must be provisioned with privileges as they start up. Automated services cannot wait around for users to type in passwords or provide credentials! So we need new agile and automated techniques to provision data, identity, and access rights.

Use Cases

There are several reasons secrets management is needed and use cases are diverse. That said, the key issue to address is to securely provision access rights to services, which is largely absent today. Instead, secrets are typically kept in cleartext within documents and applications. These issues become more pressing as large enterprise adopt these more agile deployment methods, and do so at scale. Most firms already rely upon identity stores systems to maintain a central point of control over identity and access rights. What they lack is the distribution mechanism to consistently support security policies across mixed/complex environments found at large enterprises, both on-premises and public cloud environments.

Principal Customer Use Cases

1. **API Gateways and Access Keys:** Application Programming Interfaces are how software programs interact with other software and services. These API form the basic interface for coordinated operation. To use an API you must first authenticate yourself — or your code — to an API gateway. This is typically handled by providing an access key, token, or response to a cryptographic challenge. For ease of automation many developers hard-code access keys, leaving themselves vulnerable to simple file or code inspection. And all too often, even kept in a private file on a developer's desktop, keys are accidentally shared or posted — sometimes to public code repositories. The goal here is to keep access keys secret, while still provisioning them to valid applications as needed.

2. **Services:** Applications are seldom stand-alone entities. They are typically comprised of many different components, databases, and supporting services. With current application architectures, we launch many instances of an application to ensure scalability and resiliency. As we launch applications, whether in containers or atop virtual machines or servers, we must provision them with configuration data, identity certificates, and tokens. How does a newly created virtual machine, container, or application discover its identity and access the resources it needs? How can we uniquely identify a specific container instance among a sea of clones? In the race to fully automate their environments, organizations have automated so fast they tend to get out over their skis, with little security and a decided imbalance towards build speed. Developers typically place credentials in configuration files which are conveniently available to applications and servers on startup. We find production credentials shared with quality assurance and developer systems, which are often much less secure and unmonitored. They are also frequently

shared with other applications and services which should not have access. The goal is to segregate credentials without causing breakage or unacceptable barriers.

3. **Build Automation:** Most software build environments are insecure. Developers feel security within development slows them down, so they often bypass security controls in development processes. Build environments are normally under developer control, on development-owned servers, so few outsiders know where they are or how they operate. Nightly build servers have been around for over a decade, with steadily increasing automation to improve agility. As things speed up, we remove human oversight. As new code, formation templates, and scripts are checked into repositories, build servers like Jenkins and Bamboo automatically regenerate applications. But they also validate new additions running quality assurance and security tests, halting the process should these tests fail. This means quality tests and security tests are part of the build process, and no longer slow development process, rather act as a safeguard so that bad code no longer makes it into production. Provisioning of secrets into the process, because it is also part of the automation process, is just as agile and automatic as building and launching applications. We define, through policies, the rights to be granted, ensuring we deliver code and services securely and without need for human intervention.

4. **Provisioning Machine Identities:** As we leverage cloud services our definition of 'server' changes. The cloud uses the concepts of compute, network, and storage as services to be allocated as needed and retired when not. So what we previously thought of as on-premise 'machines' have become ephemeral instances of a machine, often both virtual and multiple; represented as a server image, a container, or some similar concept. But because we fluidly bring these 'machines' up and down, it has become very difficult to keep track of which server responded to which request, so we need some way to issue them unique identities. We might need to find a potentially unhealthy or compromised machine instance, and conduct incident response, but we cannot blindly take the same remediation action against ***every*** instance, so targeting a unique identity is critical.

5. **Encrypted Data:** Providing encryption keys to unlock encrypted volumes and file stores is a common task, both on-premise and for cloud services. Traditionally we used key management servers designed to handle secure distribution and management of keys, but a number of commercial key management tools (both hardware and software) have not yet been augmented for Infrastructure or Platform as a Service. Additionally, developers now demand better API integration for seamless use with applications. This capability is frequently lacking, so some teams use cloud-native key management, while others opt for secrets management as a replacement.

6. **Sharing:** Collaboration software has helped development, quality assurance, and product management teams cooperate on projects — even though people work remotely or enterprises with teams on different continents. In some contexts, the issue is how to securely share information across a team of remote developers, or share secret data across multiple data centers without exposing it in cleartext. The databases that hold data for chat and collaboration

services tend not to be very secure, and texting certificates to a co-worker is a non-starter. The solution is a robust central repository, where a select group of users can store and retrieve secrets.

Of course there are plenty more use cases. In interviews, we discussed everything from simple passwords to bitcoin wallets. But for this research we needed to focus on the issues developers and IT security folks asked about.

Features and Functions

It's time to discuss the core features of a secrets management platform. As this is a new class of product there is little feature consistency in the market. Many are graciously referred to as 'products' and are little more than personal productivity tools. Full-function, enterprise class platforms are the exception. However, there are basic functions every secrets management platform needs to address; secure storage of secrets, provisioning secrets, identity management, and API access to aid programatic integration. When considering what you need from such a platform the key to keep in mind is that most of them were originally developed to perform a single very specific task — such as injecting secrets into containers at runtime, or integrating tightly with a Jenkins build server, or supplementing a cloud identity service. Those do one thing well, but rarely address multiple use cases and create unmanaged security islands.

Now let's take a closer look at key features.

Core Features

Secrets management platforms are software applications designed to support other applications with a very important task: securely storing and passing secrets to the correct parties. The most important characteristic of a secrets management platform is that ***it must never leave secret information sitting around in cleartext!*** Secure storage is Job #1.

Storing Secrets

Almost every tool we reviewed provides one or more encrypted repositories — which many products call a 'vault' — to store secret information. As you insert or update secrets in the repository, they are automatically encrypted prior to being written to storage. Shocking though it may be, at least one product we reviewed does not actually ***encrypt*** secrets — instead secrets are stores in plain-text. This should disqualify it from consideration. Fortunately most vaults use vetted implementations of well-known algorithms to encrypt secrets. But it is worth vetting any implementation, with your regulatory and contractual requirements in mind, to ensure the vault meets your security requirements.

With the exception of select platforms which provide 'ephemeral secrets' (more on these later), all secret data is stored within these repositories for future use. Nothing is stored in cleartext. How each platform associates secrets with a given user identifier, credential, or role varies widely. Each platform has its own way of managing secrets internally, but they typically use a unique identifier or key-value

pair to identify each secret. Some store multiple versions of each secret, so changes over time can be recalled if necessary for recovery or auditing, but the details are part of their secret sauce.

Repository structures vary widely between offerings. Some store data in simple text or JSON files. Some use key-value pairs in a NoSQL style database. Others use a relational or NoSQL database. A couple employ multiple repository types to increase isolation between secrets and/or use cases. The repository architecture is seldom determined primarily by strong security — more influential drivers include low cost and ease of use for product developers. And while a repository of any type can be secured, the choice of repository impacts scalability, how replication is performed, and how quickly you can find and provision secrets.

Another consideration is which data types a repository can handle. Most platforms we reviewed can handle any type of data you want to store: string values, text fields, N-tuple pairings, and binary data. Indexing is often performed automatically as you insert items, to speed lookup and retrieval later. Some platforms really only handle strings, which simplifies API but limits usability. Again, products tailored to a particular use case may be unsuitable for other uses or across teams.

Identity and Access Management

Most secrets management platforms concede IAM to external Active Directory or LDAP services, which makes sense because most firms already have an IAM infrastructure in place. Users authenticate to the directory store to gain access, and the server leverages existing roles to determine which functions and secrets the user is authorized to access. Most platforms are also able to use a third-party Cloud Identity Service or Privileged Access Management service, or to directly integrate with cloud-native directory services.

Interfaces and Usage

Most platforms provide one or more programming interfaces. The most common for serving secrets in automated environments is an access API. A small and simple set of API calls is provided to authenticate a session, insert a record, locate a secret, and share a secret to a specific user or service. More advanced solutions also offer API access to advanced or administrative functions. Command-line access is also common, leveraging the same basic functions in a command-driven UNIX/Linux environment. A handful of tools also offer a graphical user interface, either directly or indirectly, sometimes through another open source project.

Sharing Secrets

The most interesting aspect of a secrets management system is how it shares secrets with users, services, or applications. How do you securely provide a secret to its intended recipient? How do you establish a chain of trust from the identity store to services which need secrets? As with repositories, discussed above, secrets in transit must be protected — which usually means encryption. There are many different ways to pass secrets around securely with mutual authentication. Let's review the common methods.

- **Encrypted Network Communications:** Authenticated services or users are passed secrets, often in cleartext, within an encrypted session. Some use Secure Sockets Layer (SSL), which is not

ideal, and we recommend avoiding those platforms if possible. Thankfully most use current versions of Transport Layer Encryption, which offers bi-directional authentication between the recipient and the secrets management server. When leveraging TLS these platforms build in basic Certificate Authority capabilities. While not a full-blown PKI server; their internal CA means they can create, issue, and revoke certificates; and therefore act as the central authority on cryptographic identities *within* a secret sharing system (***e.g.:*** Docker Swarm, Kubernetes Pods, cloud autoscale group, etc.).

- **PKI:** Several secrets management platforms combine external identity management with a Public Key Infrastructure to validate recipients of secrets and transmit PKI encrypted payloads. The platform determines who will receive a secret, and then encrypts the content with the recipient's public key. This ensures that only the intended recipient can decrypt the secret using their private key. It also lessens or entirely removes the need to encrypt all network communications.

- **Temporary Files:** When using containers it is common to share secrets by placing them in a memory-only filesystem, `tmpfs` in UNIX parlance. This enables a secrets management server to provision secrets to containers hosted on the same hardware. Access to this data is limited to applications on a single physical system. Because the data is stored only in memory, access is very fast and secrets disappear when the server is de-provisioned. The downside is that such secrets are often stored in cleartext, so great care must be taken to launch only authorized containers and to configure the namespace to prevent unauthorized applications from reading them. If malicious code is introduced to any container on the physical host, this model falls apart.

- **Wrapping:** Some commercial platforms and cloud vendors use symmetric key encryption natively, with a *new and unique* key provisioned each time a service or agent is initialized. Similarly to the PKI scenario, secrets are encrypted — or wrapped — on demand with the recipient's key, then transmitted as encrypted values. The key is ephemeral, just like the cloud service, and discarded when the agent or service is terminated.

- **Injection:** In some cases secrets are provided automatically. When launching a virtual server the secret might be a configuration file provided at launch. Containers may be supplied an identity certificate which grants access and privileges within a swarm or pod. In this model each container in a pod or Kubelet could share the same set of secrets. The goal is to mitigate the risk of rogue code entering an environment and automatically gaining access to secrets.

- **Cleartext:** Yes, unencrypted plain-text. We **seriously** cannot recommended secrets management systems which fail to protect secrets, instead sharing them as plain-text. For most organizations this is a nonstarter, because it prevents them from ensuring secrets stay secret. But you need to be aware this still happens — both as secrets are moved over the network, and shared in a common directory or file, under the dangerous assumption the directory or file is inherently secure. If you see this find a different product.

Advanced Features

As the need for secrets management evolved, we began to see commercial secrets management products. These platforms are architected to support several of the major use cases discussed earlier, and typically offer more advanced features such as deep log creation and integration options, tighter integration with IAM services, secret generation, and secret revocation. As this segment matures we are beginning to see more advanced feature sets and better service integration so you need less glue code. Below is a list, in no particular order, of advanced features we have come across.

- **Managing and Administering Secrets**: For any secrets management platform the concept of the authorization model comes into play. Identity Management and directory services typically act as the 'system of record' on what identities are allowed to read, or modify, update or delete identities. One aspect that differentiates enterprise class tools is the ability to delegate a management role for administering secrets and secrets access. Logging, storage, secret creation, recovery, and failover settings are becoming table stakes for corporate secrets management platforms, and should be accessible only through the management interface — not exposed to general system users. This is an important component as it provides the ability to set security policies on usage, often a required element for regulatory or internal risk and compliance policy (*e.g.*: Rotation, separation of duties, etc). The ability for a machine or service to have the function of reading from storage, but not able to write or delete, would be one example. As we rely more and more upon automated services, it is more and more likely that the administration of secrets management will be held between humans and automated services, creating, updating and enforcing rules.

- **Provisioning Machine Identities:** Automation is the fundamental reason secrets management is now an essential feature in development and IT environments. They must be able to securely bootstrap a service, container, or server, and identify it precisely within an otherwise identical group. Shocking though it may be, some products cannot provide this basic feature, or require secrets to be dropped into a shared file system, rather than issuing a unique secret directly to each machine or instance. You will want to carefully examine how machine identities are created and provisioned.

- **Secrets Creation:** Secrets management platforms are now capable of creating and issuing SSL certificates, passwords, TLS certificates, identity tokens, encryption keys, and other useful items. In some cases these secrets can have a 'sell-by' date, after which the secret is no longer valid, for short-term access.

- **Revocation:** This enables a secrets management system to retire or invalidate a certificate. This feature is commonly available in systems where the secrets manager also acts as an identity store

or Certificate Authority — such as in container orchestration environments — and therefore can revoke a client's ability to communicate with other users and services.

- **Ephemeral Secrets:** Things like containers, servers, and IaaS/PaaS functions are essentially ephemeral. Resiliency is provided by launching many instances of an application, simply replacing any which become unhealthy. This concept works for security as well, with the idea that provisioned secrets can be just as ephemeral as a cloud server. We can generate new ephemeral secrets for server instances or container classes as needed. If a secret is lost or a container fails we generate a new secret on demand. This is useful for identity certificates, encryption keys, and other types of secrets shared between several services. It's also conducive to secrets and key rotation to aid in compliance requirements. These secrets are not stored long-term — instead the secrets manager keeps a dynamic list of which services have been issued which short-lived secrets.

- **Encryption as a Service:** Some secrets management platforms encrypt payloads on request. A simple API call passes the payload in with a unique identifier: either the encryption key to use or the intended recipient — the secrets management platform serves as an encryption engine. This relieves developers from worrying about encryption libraries, random number generation, or other encryption esoterica. As we see more encryption vendors move into secrets management, expect to see significant overlap between key management and secrets management.

- **Audit Logs:** In this day and age if you want to sell security software to enterprises, you had better offer audit logs. More and more platforms offer log files today, and some even offer `syslog` and/or JSON formats. The quality of the content and filtering remain issues in many cases, but we have reached a point where most secrets management tools include logging, at least. However, not all solutions offer the ability to store audit logs in an immutable vault where they cannot be altered or accessed by unauthorized parties which is an important security requirement for most organizations.

- **Proxy Access:** The line between Privileged Account Management (PAM) security and secrets management is beginning to blur. This capability essentially means that a secrets management service keeps access credentials secret, but provides a token (or role, in Amazon Web Services parlance) to authorize a requesting entity.

We list all these features to help readers seeking to address specific use cases. Our goal is to help you understand the available capabilities and how they can help address your needs while satisfying your IT security requirements. We also want to help you understand why certain products work the way they do, and provide an idea of what to expect from the market.

Deployment Considerations

We will close with a look at operational considerations for selecting a secrets management platform. Rather than a giant survey of products and how each works, we will focus on the facets which enable them to handle our use cases. Central questions include how these platforms deploy, how they provide scalability and resiliency, and how they integrate with the services they supply secrets to. To distinguish between products you need to understand why they were created, because core functions and deployment models are heavily influenced by each platform's intended use.

Classes of Products

Secrets management platforms fall into two basic categories: general-purpose and single-purpose. General-purpose solutions can provide secrets of many types for multiple use cases. They can automatically provision secrets to just about any type of application — from sending username and password to a web page, to issuing API keys, to dynamic cloud workloads. Single-purpose options — commonly called 'embedded or native' because they install into another platform — are typically focused on a single use case. For example several embedded solutions focus on provisioning secrets to Docker containers, nesting into your orchestration manager (***e.g.***: Swarm, Kubernetes, DC/OS), etc.

This distinction is critical because a product embedded into a container manager may not be suitable for non-container use cases. The good news is that many services are deployed this way, so embedded tools are still useful in many environments, and because they leverage existing infrastructure they tend to integrate well and scale easily. These platforms typically leverage specific constructs of their orchestration manager or container environment to provide secrets. They also tend to make assumptions about how secrets are used — for example they might leverage Kubernetes' namespace to enforce policy or the UNIX namespace to distribute secrets. Because containers are ephemeral, ephemeral or 'dynamic' secrets are often preferred for these secrets managers. The bad news is that some embedded tools assume your cluster is a secure environment, within which they can safely pass and store secrets in cleartext. Other embedded tools fully encrypt secrets, but may not support diverse types of secrets or integrate with non-containerized applications.

A product focused on a single use case may be what you need, but keep in mind that automation occurs across many different facets of development and IT, and it may be limiting. General-purpose products are typically more flexible and may take more time and to effort set up, but provide a breadth of functions not generally found in tools created specifically for container orchestration or password management.

Deployment Models

Solitary Servers

Common among early tools focused on personal productivity, solitary servers are exactly what their name implies. They typically consist of a central secret storage database and a single server instance that manages it. Basically all functions — including user interfaces, storage management, key management, authentication, and policy management — are handled by a single service. These tools are commonly used via command-line interfaces or API, and work best for a small number of systems.

Client-Server Architecture

The label for this model varies from vendor to vendor. Primary/Secondary, Manager/Worker, Master/Slave, and Service/Agent are just some terms to describe the hierarchical relationship between the principal service which manages the repository of secrets, and the client which works with calling applications. This is by far the most common architecture. There is a repository where encrypted secrets are stored, usually a database which is shared or replicated across one or more manager nodes. Each manager can work with one or more agents to support their service or application.

This architecture helps provide scalability and reliability by spawning new clients and servers as needed. These products often deploy each component as a container, leveraging the same infrastructure as the applications they serve. Many embedded products use this model to scale.

When evaluating solutions based on a client-server architecture, it is important to understand how each solution handles high availability and disaster recovery since solutions vary on their ability to handle various failure scenarios and the degree of architectural complexity.

We discussed earlier how secrets are shared between a secrets management tool and a recipient, whether human or machine. And we covered integration with container management and orchestration systems, as many tools were designed to do. It's time to mention the other common integration points and how each works. Note that solutions vary in terms of the amount of effort or glue code required to integrate the secrets management platform with various systems.

- **Build Servers:** Tools like Jenkins and Bamboo are used by software development teams to automate building and verification of new code. These tools commonly access one or more repositories to get updated code, grab automation scripts and libraries to set up new

environments, connect to virtual or cloud services to run tests, and sign code before moving tested code into another repository or container registry. Each action requires specific credentials before it can take place. Secrets management integration is either performed as a plug-in component of the build server or as an external service with which it communicates.

- **IT Automation:** Automated builds and the power of build managers have vastly improved development productivity, but orchestration tools are what move code at warp speed from developer desktops into production. Chef/Puppet/Ansible are the trio of popular orchestration tools automating IT and development tasks, the backbone of Continuous Integration and Continuous Deployment. Virtually any programmable IT operation can be performed with these tools, including most VMware and all cloud services functions offered through API. As with build servers, secrets management typically installs as a component or add-on module of the orchestration tool, or runs as a service.

- **Public Cloud Support:** The public cloud is a special case. Conceptually, ***every*** use case outlined in this series applies to cloud services. And because every service in a public cloud is API-enabled, it is the ideal playground for secrets management tools. What's special about cloud services is how integration is managed: most secrets management tools which support the cloud directly integrate with either cloud-native identity systems, cloud-native key management, or both. This offers advantages because secrets can then be provisioned in any region, to any supported service within that region, leveraging existing identities. The cloud service can fully define which users can access which secrets. Secrets management can then augment both security and compliance by placing additional usage policies on secrets, or wrapping them in another layer of encryption. There are also cases where customers ***do not want*** full integration with their cloud services, preferring to keep certain secrets and encryption keys out of the hands of their cloud service vendor so the vendor cannot be compelled to turn them over by court order. The other downside of implementing a cloud-specific solution is not having the flexibility of changing cloud providers.

Summary

As we leverage cloud services and rely more heavily on automation to provision applications and IT resources, we find more and more need to securely get secrets to applications and scripts. The need for Secrets Management has been born out of the need to automate and orchestrate IT and applications without humans to provide credentials. Developers are aware that encryption keys and API certificates sit unprotected on disk, but their focus is on delivering code faster and more easily. It is time to make it safer too. Secrets Management tools can solve the problem, and fit the environments where secrets are needed. They include API to enable inclusion in scripts and automated services, fitting perfectly within a DevOps operational model.

If you have any questions on this topic, or want to discuss your situation specifically, feel free to send us a note at info@securosis.com or post a question on our blog.

About the Analyst

Adrian Lane, Analyst/CTO

Adrian Lane is a Senior Security Strategist with 25 years of industry experience. He brings over a decade of C-level executive expertise to the Securosis team. Mr. Lane specializes in database security, secure application development and data security. With extensive experience as a member of the vendor community (including positions at Ingres and Oracle), in addition to time as an IT customer in the CIO role, Adrian brings a business-oriented perspective to security implementations. Prior to joining Securosis, Adrian was CTO at database security firm IPLocks, Vice President of Engineering at Touchpoint, and CTO of the secure payment and digital rights management firm Transactor/Brodia. Adrian also blogs for Dark Reading and is a regular contributor to Information Security Magazine. Mr. Lane is a Computer Science graduate of the University of California at Berkeley with post-graduate work in operating systems at Stanford University.

About Securosis

Securosis, LLC is an independent research and analysis firm dedicated to thought leadership, objectivity, and transparency. Our analysts have all held executive level positions and are dedicated to providing high-value, pragmatic advisory services. Our services include:

- **The Securosis Nexus**: The Securosis Nexus is an online environment to help you get your job done better and faster. It provides pragmatic research on security topics that tells you exactly what you need to know, backed with industry-leading expert advice to answer your questions. The Nexus was designed to be fast and easy to use, and to get you the information you need as quickly as possible. Access it at <https://nexus.securosis.com/>.

- **Primary research publishing**: We currently release the vast majority of our research for free through our blog, and archive it in our Research Library. Most of these research documents can be sponsored for distribution on an annual basis. All published materials and presentations meet our strict objectivity requirements and conform to our Totally Transparent Research policy.

- **Research products and strategic advisory services for end users**: Securosis will be introducing a line of research products and inquiry-based subscription services designed to assist end user organizations in accelerating project and program success. Additional advisory projects are also available, including product selection assistance, technology and architecture strategy, education, security management evaluations, and risk assessment.

- **Retainer services for vendors**: Although we will accept briefings from anyone, some vendors opt for a tighter, ongoing relationship. We offer a number of flexible retainer packages. Services available as part of a retainer package include market and product analysis and strategy, technology guidance, product evaluation, and merger and acquisition assessment. Even with paid clients, we maintain our strict objectivity and confidentiality requirements. More information on our retainer services (PDF) is available.

- **External speaking and editorial**: Securosis analysts frequently speak at industry events, give online presentations, and write and/or speak for a variety of publications and media.

- **Other expert services**: Securosis analysts are available for other services as well, including Strategic Advisory Days, Strategy Consulting engagements, and Investor Services. These tend to be customized to meet a client's particular requirements.

Our clients range from stealth startups to some of the best known technology vendors and end users. Clients include large financial institutions, institutional investors, mid-sized enterprises, and major security vendors.

Additionally, Securosis partners with security testing labs to provide unique product evaluations that combine in-depth technical analysis with high-level product, architecture, and market analysis. For more information about Securosis, visit our website: <http://securosis.com/>.