Degree project

Secure Introduction for Enterprise Secrets: An Evaluation Framework

Master Degree Project (120 credits) in Informatics with a specialization in Privacy, Information Security and Cyber Security

Second Cycle 30 credits

Spring term 2021

Student: Ulf Weltman

Supervisor: Yacine Atif

Examiner: Jianguo Ding

Acknowledgements

I would like to express my great appreciation to my supervisor, Yacine Atif, for his valuable advice throughout this research work, and to my examiner, Jianguo Ding, for his excellent feedback.

Also, my profound gratitude to Lucy and Lyly Weltman for their daily support and encouragement during this research, and to Rob and Helena Weltman for the opportunities and experiences that made me the person I am.

Abstract

A dependency on secrets is inherent in most IT systems, especially as they become increasingly complex and interdependent. Vast amounts of research have explored how to protect the confidentiality, integrity and authenticity of secrets through means such as encryption and authentication. These means are in themselves supported by secrets, and introducing those secrets is an area that has seen less exploration. Secrets are protected by secrets, and the secret at the top needs to be provided by one of the numerous methods with various advantages and disadvantages. This work follows a design science research approach to design a framework for comparing those methods of secure introduction, demonstrated through scenarios and practical exercises.

Keywords: secure introduction, enterprise, secrets, framework, design science research

Table of Contents

1 Introduction

Enterprise information technology systems depend on secrets to support internal interaction between the components that form the systems. An illustrative example is an online shop consisting of a web application and a database. The database contains customer information so it is likely protected. The web application must authenticate itself to the database, and the database software might encrypt the customer information when it is not in use. The secrets in this simple case are a database credential used by the web application and a cryptographic key used by the database software. These secrets are specialized information that serves to support the primary information flows that are processed by the system.

Enterprise IT system secrets are not limited to credentials or cryptographic keys and could include other types of supporting information. A unifying attribute of the secrets is, as the name suggests, that they are not public information. They support processes that could be interfered with, either accidentally by a privileged user or intentionally by a malicious user, and the secrets must therefore be protected. At the same time the IT systems cannot function if the secrets are not available so the secrets need to be introduced to the systems in some secure way. There are several methods of providing secure introduction of secrets to enterprise IT systems and each comes with some advantages and disadvantages. These differences between the methods of secure introduction may not be obvious as they are not standardized or even discussed at a comparative level either in academia or application space at the time of this writing.

This work explores how methods of secure introduction of secrets for enterprise IT systems can be evaluated in comparison to each other. This will be done through design science research drawing on the work by Peffers et al. (2006). The paper's structure follows their model for design science research, presented as their six process steps: problem identification, objectives of a solution, design and development, demonstration, evaluation, and communication.

2 Background

The concept of secure introduction of secrets is as old as information systems, but searches in academic and web search engines for related terms reveal that implementations in the enterprise have not been thoroughly explored by academia at the time of this writing. However, the discrete processes that form methods of secure introduction are well traveled in academia. Securely storing secrets and accessing them through system authentication and automation form the gestalt of secure introduction of secrets in IT systems.

2.1 Secrets

Cryptography is used to protect the confidentiality of information. Encrypting information at rest is a requirement of some standards such as NIST SP 800-53 control SC-28(1) (National Institute of Standards and Technology, 2020), requiring the “information system implements cryptographic mechanisms to prevent unauthorized disclosure.” The confidentiality of a system’s primary information, e.g. customer records and sale transactions, benefits from being encrypted at rest to prevent their disclosure to unauthorized parties. The cryptographic keys used for encryption and decryption are considered secrets and need to be protected because access to the keys implies access to the protected primary information. Therefore cryptographic keys and service credentials can also be encrypted at rest and saved in a secrets store, conceptually similar to a human’s password manager. Similar to a password manager, access to the secrets store also needs it to be unlocked before the secrets can be retrieved to support the system’s processes. The key that unlocks the secure store cannot reside in the secrets store because that would result in a cyclic dependency. The key to the secrets store needs to be introduced to the system before its main processes can start.

2.2 Automation

In the example of the web application and the database introduced in the Background chapter it would be possible for an operator to manually enter the key required to unlock the secrets store before the system starts its processes. However, using a human to introduce secrets is inefficient and risky. Humans have biological needs such as sleep, lower reaction times than computers, the need to memorize keys, and so on. This adds toil to the system which is work that is “manual, repetitive, automatable, tactical, devoid of enduring value, and scales linearly as a service grows" (Beyer et al., 2016). Automation can reduce toil at various levels of a system to reduce the human requirement. We can consider automation at various levels of a system. Three levels considered in this work are the levels of infrastructure management, the software configuration, and the system runtime.

The lifecycle of a system’s infrastructure can be automated using Infrastructure-as-Code. IaC defines the infrastructure that the system runs on as a “blueprint that contains deployment specifications” (Matej Artač et al., 2017). This way a predefined set of computing, storage, networking and other components can be created, modified or removed when the blueprint is executed. Human operator error and latencies are reduced.

The configuration of the system can be automated using Configuration-as-Code. “CaC is the technique of defining computing and network configurations through source code” (Rahman et al., 2018), bringing to service configurations the same benefits as IaC brought to infrastructure, improving the correctness and timeliness of the configurations.

At runtime the system’s resources can be managed using platform specific features. A bare metal server (Yeoman, 2019) can include some specialized

hardware such as a Hardware Security Module that offloads cryptographic functions and protects private keys (Fox, 2009). A virtualized platform such as virtual machines in public clouds can be provided with identities that can be authenticated by the cloud provider application programming interface (API) ("Amazon Elastic Compute Cloud," 2021; "Google Compute Engine," 2021). Similarly, the Kubernetes system for container lifecycle management provides identities for virtualized applications ("Kubernetes Authentication," 2021). In both cases the hypervisor or controller is aware of the managed tenants and can therefore insert a credential that identifies the tenant.

2.3 Secure introduction

It could be argued that the concept of secure introduction of secrets is as old as information systems, or much older if we stretch the definition to include archeological artefacts such as the keys to ancient paper ciphers. To limit the scope, in recent years the application of secure introduction of secrets in enterprise systems has become a hot topic, brought on by increasing scale and complexity of operations and the threat of ever escalating cybersecurity attacks.

As the scale and complexity of systems grow, the secrets they require to support their processes will naturally increase in number and distribution as well. Unless the same secret is used across the system, which would be akin to a human using the same password everywhere, the number of distinct secrets grows at some pace along with the complexity of the system. As the overall systems scale out horizontally, the secrets need to be distributed to more instances of the subsystems, with each system depending on some subset of the overall secrets of the system. Managing the lifecycle of the systems though continuous integration and delivery practices can result in a more agile operation as systems can be updated more frequently (Humble and Farley, 2010), but then systems will need to be recreated or restarted more frequently and without human intervention. The need for automated secure introduction increases.

A simple method of managing secrets is to store them in decentralized plaintext files, such as configuration files, on each server. This implies some trust in operating system protections so that only authorized parties are able to access the file. However, the network security architecture Zero Trust (Kindervag et al., 2016) originally introduced by Forrester Research market research company has been making strides in the industry and standards, including NIST SP 800-207 (2020), which states "Zero trust assumes there is no implicit trust granted to assets or user accounts based solely on their physical or network location (i.e., local area networks versus the internet) or based on asset ownership (enterprise or personally owned)." In the context of information system secrets this suggests that having access to a host should not imply having access to the secrets used by the services on the host. A practical example is that of different operational teams such as system administrators tasked with keeping hosts running, and service administrators tasked with keeping the service running. The system administrator requires

privileged access to the hosts to debug any problems such as hardware and operating system issues while the service administrator requires privileged access to manage the service's secrets. These two administrators' privileges do not intersect, and per the principle of least privilege "Every program and every privileged user of the system should operate using the least amount of privilege necessary to complete the job" (Saltzer, 1974).

Protection of information has been thoroughly explored by academia and the application space. There have been large amounts of research into security aspects such as cryptography, communications, authentication and password management. This has resulted in strong ciphers such as AES, secured protocols such as TLS, access control systems and recommendations in password management. The information protected by these mechanisms can be considered relatively secure after all the effort to improve them. And yet, there is a gap at the start of the processes where some plaintext secret is required for authentication or decryption and starting the chain of accessing secrets. The strong protections of transport encryption and access control are circumvented if this initial secret is not introduced securely. The initial secret is the weakest link.

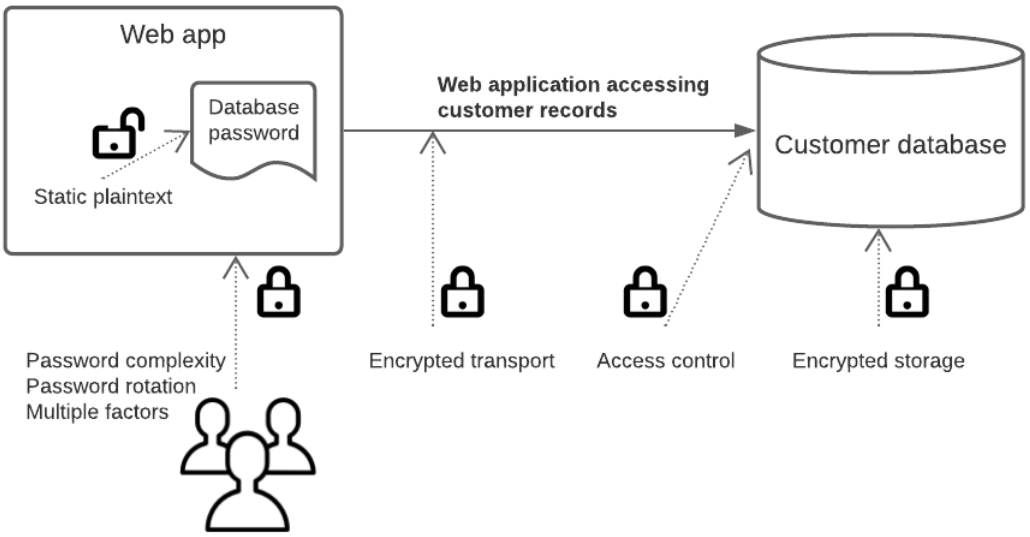

Figure 1: The insecurely introduced secret as the weakest link

2.4 Existing methods

There are several methods of introducing secrets to accommodate the differing needs of organizations and the differences in the platforms underlying their systems. These methods do not offer a uniform level of protection and they vary in the toil and complexity to configure and maintain them. Additionally, some of the methods have dependencies on features provided by particular platform types. Three high level examples to highlight these differences are managed platforms, and unmanaged platforms with or without configuration management and hardware support. These three are examples selected from literature to highlight how each is distinctly different from the others. This is

not a complete list.

In these examples the entities of web app and customer database are remote, separated by network. They are in many-to-many relationships meaning that there are multiple web apps connecting to multiple databases, although the diagrams show a single pair for brevity. The web app and customer database entities are in many-to-one relationships with the secrets store, representing centralized secrets management. The data transfers are depicted with a solid arrowhead for remote connections or a hollow arrowhead for local invocations.

In this work the focus is on the authentication to the secrets store. This is a simplification of the system. In each of the following example figures the "get secrets" and "use secrets" data exchanges are depicted for context but are not in scope for evaluating the framework attributes. These exchanges could be HTTP requests and database connections that are secured using common practice techniques such as encryption and access controls.

2.4.1 Example: Managed platforms

Managed platforms provide some form of multi-tenancy to allow multiple systems to run in isolation on the same physical hardware. These managed platforms provide a controlling mechanism to manage the lifecycle of the tenants. A feature of some of these controllers is that they can dynamically add some information that represents the identity of the tenant and then provide authentication of those tenant identities as mentioned in the previous Automation section. If the secrets store, viewed as a third party, is configured to trust the controller's tenant authentication then this can be used as a method of secure introduction as depicted in Figure 2 which is a visualization based on literature provided by various vendors of managed platforms ("Amazon Elastic Compute Cloud," 2021; "Google Compute Engine," 2021; "Kubernetes Authentication," 2021)

The identity may concretely be a signed JSON Web Token (JWT) which is a set of data that encodes some optional claims about the holder (Bradley et al., 2015). These claims may include the subject (sub), an expiration time (exp), as well as other standardized or customized application-specific claims. Importantly, the JWT is also cryptographically signed to allow it to be authenticated. The identity in step 1 of Figure 2 can therefore be a JWT that represents the application's identity. In step 2 the JWT is transmitted to the secrets store. In step 3 the JWT is passed along to the controller to verify that the expiration time of the JWT has not passed and that the cryptographic signature matches the controller's public key. Any other application-specific claims such as whether the JWT matches the expected geographical region of the controller can also be checked.

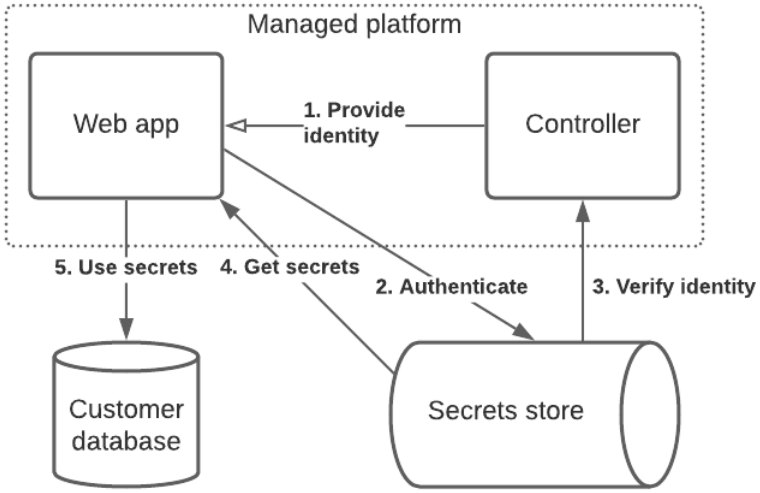

Figure 2: Managed platform

2.4.2 Example: Unmanaged platforms with configuration management and HSM

Unmanaged systems lack the automatic identity management provided by a managed platform. Instead they can be configured by a configuration management tool that creates and maintains the system configuration at an application level, dynamically inserting values into configuration files. An example of this would be a bare metal server hosting a single tenant (Yeoman, 2019), configured by a tool such as Puppet (Rahman et al., 2018). The configurations are version controlled by some controller hosts. The controller hosts need to protect the keys used to sign the secrets for the configured hosts, so controllers can have specialized hardware to provide the configured cryptographic identities that can be authenticated by the secrets store. The configuration controller hosts can therefore use a Hardware Security Module (HSM) to securely store private keys and provide signing operations without disclosing the keys (Fox, 2009). In this scenario the HSM can sign an identity construct including a JWT or a X.509 key pair (Cooper et al., 2008) on behalf of the configured host, and the secrets store can be configured to trust the public key associated with the signature, as depicted in Figure 3. In this case the secure store has been preconfigured to trust the HSM's public key and performs the verification by itself, aside from checking a revocation list to determine whether the client's certificate is still valid.

Figure 3 is a visual representation of configuration management literature. Each of the boxes in the figure is a distinct remote entity except for the HSM which may by an expansion card installed into the configuration manager host.

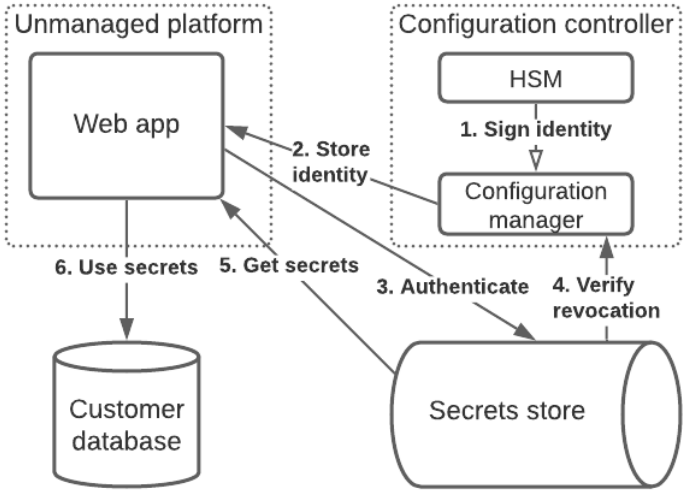

Figure 3: Unmanaged platform with configuration management

2.4.3 Example: Unmanaged platforms without hardware support

An unmanaged platform without hardware to provide a cryptographic identity relies on software constructs of its own. In this case the credential resides on persistent storage instead of being an intrinsic identity provided by a controller or hardware. The credential can be encrypted, but then another decryption key is required to be present to allow the credential to be decrypted before it is ready to be used.

An example of this is passwords such as database credentials stored in application configuration files. Another is the private key for a TLS server or client certificate saved either in a text file, or encrypted but with the decryption password in a text file ("NGINX," 2021). Given the lack of controller and continuous configuration management, these are static secrets that are provisioned with a system image along with the software when the host is installed. They are long term secrets that exist until the host is decommissioned.

Figure 4 depicts an example of a basic unmanaged platform with a static credential consisting of a X.509 certificate that was installed as part of the web app system image. The secrets store verifies that the certificate has not been revoked before granting access to the secrets.

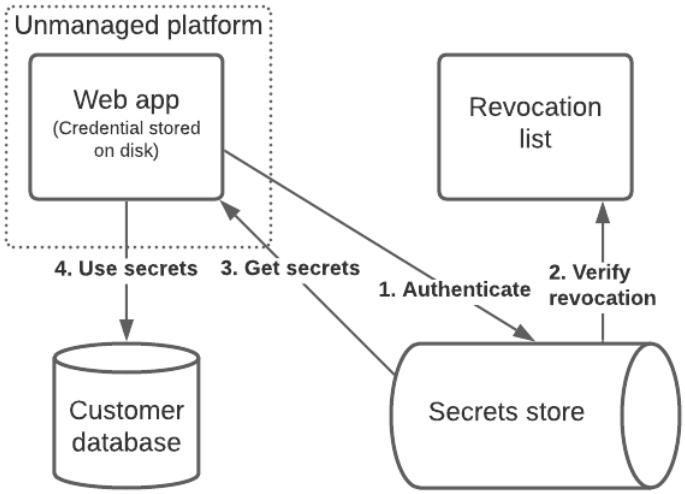

Figure 4: Unmanaged platform without hardware support

3 Problem and objectives

The need for secure introduction of secrets, and the variety in its methods, leads to the problem identification of this research. The root of the problem is related to the diversity in the available methods, but that diversity is in itself not a problem. In fact, the availability of the various methods with different implementations and strengths is beneficial in meeting the differing requirements of organizations. The variety does give rise to the problem of comparisons which leads to the following research question.

How can methods of secure introduction be compared?

There are at least two instances when it would be useful to have a systematic way of comparing methods of secure introduction.

First, selecting an appropriate method is not obvious to information system designers without a way to compare them. In most cases there will be more than one method that is compatible with a given platform and selecting a method that offers the best level of security with the least monetary and complexity cost is a choice that can have profound consequences. Being able to make comparisons would allow the information system designer to make a grounded decision.

Second, research into cybersecurity is a constant and improvements to methods of secure introduction can be expected as technologies evolve, but it is not obvious how to evaluate improvements through any new or updated methods. Being able to make comparisons would allow the secure introduction designer to evaluate the improvements.

Therefore, the main objective of this study is to design a framework for comparing methods of secure introduction of secrets in enterprise information systems.

4 Related works

While there are no published frameworks for comparing methods of secure introduction, there is a wealth of literature for comparing and evaluating software architectures in general. These may be heavy weight for the purposes of comparing secure introduction which may are only a supporting component in the overall architectures, but the architecture analysis literature may still offer some insights to be leveraged.

Software Engineering Institute (SEI) published Architecture Tradeoff Analysis Method (ATAM) which provides "a principled way to evaluate a software architecture's fitness with respect to multiple competing quality attributes" (Kazman et al., 1998). While the intent of the method is in line with what is desired for the secure introduction comparisons, such as selecting quality attributes and recognizing competing attributes, it does not actually provide the quality attributes. Those need to be tailored to the secure introduction use case.

SEI also published the business cost approach Cost Benefit Analysis Method (CBAM) for modeling economic software design decisions for architectures. It is intended to augment ATAM with business-centric tradeoff analysis, and as such is also of interest to secure introduction given varying costs of the methods. CBAM provides a relatively laborious process which may be useful for larger architectures, but too intensive for the secure introduction comparisons.

Software Architecture Comparison Analysis Method (SACAM) compares the fitness of architecture candidates by defining quality attributes and scoring them between 0 and 10 (Stoermer et al., 2003). The overall process is heavy, e.g. 14 days including workshops in an example exercise. Scoring is of interest in the secure introduction comparison framework, but the attributes being considered may not lend themselves well to quantitative scoring.

Palmius (2007) notes "the definition of what constitutes a good system varies between stakeholders, perspectives and system purposes." A method of secure introduction can be good for an organization's use case but not necessarily for others. For example, a method might fit a large enterprise but be prohibitively expensive for a startup company. Palmius' work deals with comparing whole information systems and that is much broader scope than required for this report.

There is a wealth of literature about IT security controls and risk management that ties in with secure introduction. The standards ISO/IEC 27001 (2013) and NIST SP 800-53 (2020) provide requirements and controls for information system security that an organization's secure introduction may need to comply with. Additionally, performing a risk assessment per ISO/IEC 27005 (2018) and NIST SP 800-30 (2012) before evaluating secure introduction will help to ensure a selection is appropriate given the information it protects. The controls and risk assessment can also help to establish some assumptions about trust, such as whether to place trust in an HSM to safeguard cryptographic keys.

In the application space there is documentation for various specific implementations of secure introduction to support the search for differentiators for the framework. One related vendor is HashiCorp providing the Vault product which supports several authentication methods ("Vault Auth Methods," 2021). While not providing comparative literature, examining the details of these methods reveals differences such as the requirements for external orchestration mechanisms. Another type of vendor is the cloud service providers that provide some integrated authentication mechanism such as service accounts associated with the tenants ("Amazon Elastic Compute Cloud," 2021; "Google Compute Engine," 2021; "Kubernetes Authentication," 2021).

5 Design and development

Design science research produces some artifact that is "both useful and fundamental in understanding" a problem (Hevner and Chatterjee, 2010). The aforementioned problem is that of comparing methods of secure introduction, and the objective of this research is to design a framework that defines the important attributes of comparison. These attributes are derived from the related literature, analyzing differences between the methods in the Existing methods section, and author's experiences with implementations of secure introduction. There are several angles to consider depending on stakeholder point of view. Three views included here are risk, cost, and appropriateness.

The point of secure introduction is to reduce risk by making it difficult to intercept the secrets that support an information system's processes. A general way of analyzing security concerns is through the CIA triad (Perrin, 2008). In the context of secure introduction the confidentiality of the secrets is critical as the system's resources can be exploited by an attacker with access to its secrets such as passwords and encryption keys. Unauthorized access to secrets can result in espionage, theft, privacy breaches, and so on. The availability of the secrets is also critical as the system's processes cannot function without them, so a loss of availability of the secrets can result in a costly system outage when the system cannot access its secrets store to retrieve cryptographic keys, access its customer databases, and so on. Similarly, compromising the integrity of secure introduction is also likely to result in a denial-of-service outage as an incorrect secret will result in a failure to authenticate and retrieve any secrets from the secrets store.

Another risk-related issue is that of trust. To improve the naïve method of storing secrets directly in plaintext files the secrets are moved into some other layer such as a secrets store. Some secret is still required in order to access the secrets store, so the result is a shift in the responsibility and management of the protection of the secrets. There will be some different components of the enterprise system that need to be trusted to be secure to some degree. If a cryptographic identity is provided then there is some key to decrypt it, and that key needs to be held somewhere; by a cloud service provider, in an HSM, in a human's brain, etc. It is up to the organization's risk management policy to decide which of those parts of its information system are trusted to serve the

identity to be authenticated by the secrets store. Treating the secrets management as a black box is dangerous as there may be unknown components of the system that can access or affect the access of the secrets and thus circumventing the protection. Therefore being able to enumerate the trusted components of the system is an important aspect of the evaluation.

The next view is cost which consists of two types of cost, the first of which is financial. Financial cost can be a concern in an organization such as where equipping a fleet of systems with hardware to provide a strong cryptographic identity might present a prohibitive initial capital expenditure (CapEx) cost. In other cases there might not be an initial purchase required but rather an on-going operating expenditure (OpEx) cost, such as a usage fee to a cloud service provider. There might also be zero financial cost, in either or both of CapEx and OpEx, if the platform includes free features that provide identity management for its tenants. Some methods of secure introduction have dependencies on components that incur either, both or none of the CapEx and OpEx costs, and being aware of these costs will also affect the evaluation.

Another cost-related concern is in toil, that manual and repetitive work that should ideally be automated to avoid needing to spend overwhelming amounts of the time and effort required as the system scales. The toil may be required in order to configure the system with secure introduction but especially for any ongoing maintenance to keep the system of secure introduction running when the dependent services and their requirements for secrets change. Given finite staff and competing priorities the toil required for some methods of secure introduction could present a prohibitive cost in time and effort. On the other hand, similar to the financial cost, there might instead only be a requirement for low amounts of toil if a tenant identity is a basic feature of the platform.

Finally, the appropriateness view includes that of compatibility. This includes several aspects. If a method of secure introduction requires some component or process that is not available for a given platform, such as an integrated tenant identity, then that method would be excluded from consideration unless the platform is replaced. Additionally, the organization may have functional and non-functional requirements such as compliance features, a reliable support lifecycle to address bugs and security flaws, execution speed and self-hosting or managed hosting requirements that may or may not be compatible with a given method of secure introduction. Selecting a platform has broad implications for a system beyond secure introduction, and missing other requirements such as a support lifecycle or compliance features can result in a serious gap during the evaluation.

Table 1: The five attributes of comparison for secure introduction

Attribute	**Example consideration**	**Reason for including**
Security: How well are the aspects of the CIA triad protected for the	Any user with access to the system can access the secret, or only the server process can access it.	The main reason for secure introduction is to protect the secrets.

secret?		
Trust: What are the technical and social components of the system that are given some level of trust?	The organization must trust the public cloud provider safeguarding tenant identities, or the system administrator is trusted.	Implies a possible change in attack surface and responsible parties to be considered through the organization's risk management policy.
Financial: What will the relative monetary cost be to use a method?	A dedicated HSM card per bare metal server adds some CapEx cost to each, or there might be OpEx in the form of a usage charged by a cloud platform.	Financial planning and approval might be required, particularly to equip large fleets of hosts with new hardware.
Toil: What amount of work is required to install and maintain the method?	Manual labor to install and rotate the secrets, or it is automatically provided by the platform.	A method with high toil may require so much upkeep that it is not viable in the long term.
Compatibility: Does the method provide the required features?	Bare metal server provides no controller-provided identities, but cloud platforms provide several identities.	Particular methods may not be possible to use on the given platform, or do not provide the required features.

6 Demonstration

Continuing with the design science research process of Peffers et al. (2006), this section provides demonstrations of the framework artifact.

6.1 Scenario demonstration

The Existing methods section introduced three examples: a managed platform, an unmanaged platform with HSM, and a plain unmanaged platform. Continuing with those three examples, the framework is exercised in scenarios to demonstrate how the attributes differ and how they affect the selection of those methods.

In these example scenarios fictional organizations and their needs and platforms are presented and the five attributes are judged by literature review. The details of the architecture and implementation are essential for the system

designer to evaluate the method. These are abbreviated descriptions of the systems that demonstrate the use of the framework, and some of the details are glossed over for brevity. In real life a deeper dive into details such as compliance controls is expected.

A concise first person story statement that includes the target persona and their needs and motivations is expressed to focus this literature-based demonstration.

> *As a system designer I need to evaluate a method of secure introduction for my organization so that I will not encounter unforeseen gaps.*

6.1.1 Scenario: Managed platform with Kubernetes

An organization is running a cluster of Kubernetes on a set of servers in their own private cloud. The cluster provides the automated lifecycle management of various applications running in containers. The applications are interdependent and require secrets to support their interaction.

The organization's requirements include compliance features that allow their platform to be audited for compliance with various certifications. They are also concerned about support that would address any bugs or security flaws that are uncovered in the secure introduction process.

The Kubernetes platform uses a cluster model that spans hosts with a data plane formed by a number of worker nodes and a control plane for managing the worker nodes and their hosted containers. The control plane interaction by administrators and the hosted applications goes through a component called API server ("Kubernetes Components," 2021). Processes in hosted applications interact with the API server using service accounts which are authenticated using credentials represented by JWT. Applications are deployed to logical partitions called namespaces and each namespace is assigned service accounts. In addition to the internal use, these JWT are also accessible by the applications in the containers for use in authenticating against external systems that support JWT authentication, including their secrets store. The controller automatically provides the JWT to the application running in a container. Access to the containers is restricted so external users and processes cannot enter the container to gain access to the token.

- Security: High
    - Confidentiality is well protected as the controller dynamically generates a JWT for each instance of the application, and the token is not accessible outside of the container. The tokens are only valid for the lifetime of the application instance. The validity of the claims in the token is assured, assuming trust of the controller and the controller making the claims on behalf of the container. The token is cryptographically authenticated.
    - Integrity and availability of the JWT are protected by the closed system. The main opportunity for a denial of service attack would be a deliberate misconfiguration such as breaking the

mapping between JWTs and secrets.

- Trust: Kubernetes infrastructure, Kubernetes administrators
    - Kubernetes infrastructure forms a closed system and the strength of the security relies on trusting Kubernetes to not expose the token or the private signing key. The service account and its JWT are core functions of the infrastructure.
    - Kubernetes administrators are relied on to safeguard the namespaces because a rogue container would assume the identity of its namespace and therefore gain access to its secrets. Administrator oversight is accomplished with controls such as auditing and alerting.
- Financial: None
    - CapEx and OpEx are zero as the service account JWT is a core feature of the Kubernetes platform and incurs no financial costs.
- Toil: Low
    - Low amounts of toil are required to use and maintain Kubernetes JWT authentication. This is a centralized approach requiring some reusable tooling for applications to exchange the JWT with the secrets store and for each application some configuration to map its namespace identity to a subset of the secrets store.
- Compatibility: Kubernetes platform, high auditability, good support
    - Kubernetes is the selected platform and the Kubernetes service account JWT secure introduction method is tightly integrated. Similar mechanisms exist for other managed platforms.
    - Auditability is achieved by logging and alerting. Compliance in secure introduction on this platform is focused on how applications are deployed to namespaces, and in preventing access to the containers.
    - Support is tied to the integrated identity feature as a core part of the platform operation and therefore unlikely to be neglected in terms of support. The platform is open source with fixes released for the last three minor releases ("Kubernetes Version Skew Policy," 2021).

Given the evaluation results, the system designer can conclude that the selected method of secure introduction is a good choice as it is well integrated with the platform and fills the compatibility requirements. The security is good, and using it requires no extra financial investment or significant toil. The organization is assumed to place some trust in the Kubernetes infrastructure and its administrators, but oversight of these technical and social components can be added through auditing, especially of the process for publishing applications to namespaces.

6.1.2 Scenario: Unmanaged platform with configuration management and HSM

An organization has a fleet of bare metal servers and centralized configuration management. Each server hosts a single tenant application consisting of one or more components. The applications on different sets of bare metal servers require secrets to interact with each other.

The organization is concerned about compliance as well as the execution speed because their application cannot tolerate any additional latency added by any secure introduction of its secrets.

The configuration managers have HSM installed as specialized hardware for securely storing private keys. The HSM signing keys are used in cryptographic signing operations for the X.509 certificate key pairs added to the configured hosts. The HSM take the form of expansion cards. The configuration management tool runs periodically and custom scripting created by the organization revokes and replaces the certificate X.509 certificate key pairs on the configured hosts on a cadence in order to rotate the credentials.

- Security: Medium
    - Confidentiality is achieved using the HSM to dynamically generate a client certificate key pair for each configured host and rotating them periodically. The certificates have expiration times and their subjects contain an identifier for mapping permissions. The validity of the expiration time and identifier in the certificate is assured, assuming the configuration manager is trusted, because the configuration manager creates them on behalf of the configured host. It is up to the software on the host to keep the credential safe after it has been configured as it is saved on persistent storage. The certificate is cryptographically authenticated.
    - Availability can be attacked by disrupting the periodic configuration updates that replace the certificate key pairs before they expire. If the updates are stopped then access to secrets will be denied as an expired certificate will be rejected by the secrets store. Attacking the integrity, such as corrupting the key or certificate, will also result in denial of access to secrets. Attacking the certificate revocation list by marking valid certificates as being revoked would also result in denial of access to the secrets.
- Trust: Configuration management software, HSM card, configuration administrators, system administrators
    - Configuration management software is trusted to push correct configurations to the configured hosts. Access to the configuration controllers and the configuration data needs to be limited to avoid a rogue host being configured with access to secrets.

    - HSM cards are relied on to not expose their private signing keys and to disallow unauthorized parties from performing signing operations. If the signing keys or operations can be abused then rogue certificates can be signed and secrets accessed. The HSM is designed to not allow private keys to be extracted, and it can be bound to the host, so it is not feasible to extract the key or the HSM itself to generate identities from another rogue host.
    - Configuration administrators who manage the configuration management tool are relied on to not misconfigure configuration data or misuse a super user to exercise the tool or HSM.
    - System administrators of bare metal servers need privileged access for occasional debugging and fixing when hardware or software fails.
- Financial: Medium
    - CapEx is relatively high as HSM cards are expensive, being specialized devices, and this could become a prohibitive cost for large fleets of hosts. However, centralizing them on the configuration managers limits the cost assuming the number of configuration managers is low.
    - OpEx is low assuming the configuration management software used is a free version. The configuration management software is not only used for secure introduction, so any license costs may be subsumed by the cost of configuring operating systems and applications.
- Toil: Medium
    - Toil is required to manage the components of the system. The configuration management profiles need to include resources that define identities appropriate for each set of hosts. The HSM devices also need to be managed. Without a standard for HSM devices, configuring and maintaining them, e.g. firmware updates, may present some toil. This could be alleviated with a configuration management tool for the configuration controllers as well.
- Compatibility: Broad platforms, medium auditability, fast execution
    - Broad platform compatibility is provided by this method. Provided the access, a configuration management tool can be used against physical and virtual hosts. It requires that the tenants are mutable which means that configuration of a workload in an immutable container image is not possible.
    - Auditability for compliance involves controls for the configuration manager components and auditing access to the credential stored on the configured hosts.

- Execution speed is expected to be as fast as TLS in general, using the X.509 certificates to identify the client in a TLS connection to the secrets store. The configuration controller replaces the certificate key pairs asynchronously so they are ready when the applications need them to access secrets.

Based on this evaluation the system designer can conclude that this method of secure introduction is a reasonable choice for bare metal servers. The configuration management controllers provide centralized management of the secure introduction and the HSM provides strong protection for the private signing keys. A gap may be in any interactive shell access required by system administrators with high privileges required to keep the systems running. Being able to access the certificate key pairs on disk and copying them to another rogue host would allow access to the secrets, so it is important to limit unnecessary administrators, harden the operating system and implement auditing.

6.1.3 Scenario: Unmanaged plain platform

An organization has a fleet of bare metal servers hosting their applications. Similar to the scenario in 6.1.2, each server hosts a single tenant application and the applications on different sets of bare metal servers require secrets to interact with each other. In this case each server is instantiated and then left untouched until any on-demand updates are required, such as application or operating system patches.

The organization is relatively small and concerned about compliance and the support lifecycle.

The servers are generic without any specialized hardware support for key management or cryptographic operations. They rely on a pure software approach to security, using a system image creation tool to provision a X.509 certificate key pair with long validity in each host's persistent storage when the hosts are initially installed. The certificates are part of a hierarchy that is trusted by the secure store and include subject identifiers that determine the subset of secrets allowed for each host.

- Security: Low
    - Confidentiality is poor. Although the credentials provide a cryptographic identity, they are long lived, not dynamically generated by the host, and can therefore be copied to another rogue host and used long term. Each host that is provisioned with the image gets the same credential so revoking it would disrupt multiple systems. Additionally, the credential is embedded in the image which implies that the credential is available to anyone with access to the repository where the image is stored.
    - Integrity and availability can be attacked by corrupting the certificates or keys on individual hosts. Also, if the certificate revocation list can be modified by an attacker then all the

certificates can be marked as revoked and access to secrets will be denied.

- Trust: Operating system security, image management, system administrators
    - Operating system access restrictions are important for this method to prevent unauthorized parties from gaining access to the certificate and key files on the hosts.
    - Images include the verbatim certificate key pair files and must therefore be trusted to be kept safe. Creation, storage and destruction of images are all required trusted processes due to the long lived certificates that they contain.
    - System administrators need privileged access to the bare metal servers for occasional debugging and fixing when hardware fails.
- Financial: None
    - CapEx and OpEx are zero. Although there may be some cost to an image creation tool and its certificate signing functionality, overall the solution has no financial cost for the secure introduction part of the image process.
- Toil: Low
    - Toil is required as an initial cost in automating the generation of the certificate key pairs into the process for installing new servers and the revocation of the certificates when the servers are decommissioned.
- Compatibility: Broad platforms, poor compliance, self-supported
    - Broad compatibility is provided by the pure software image approach that does not depend on a controller in a managed platform can potentially be used on many types of platforms including virtual and physical, mutable and immutable.
    - Compliance with information security standards suffers as the credentials in the form of certificate key pairs are long lived, appear in plaintext in system image files, and are shared across multiple systems.
    - Support is provided by the organization itself as it builds and runs the process. Any flaws in the process need to be resolved by the organization itself.

The system designer can conclude that this is not a good method of secure introduction in the long term. Although it is simple in concept and incurs low financial and toil costs, the security it provides is poor. There are several poor practices in this method that risk attacks on confidentiality. It is an improvement over writing credentials directly into application configuration files because of the indirection that the certificates and a secrets store provide. The certificates can be revoked and access to the secrets store audited.

6.1.4 Scenario comparisons

An overall comparison can be made after evaluating the individual methods of the scenarios. Table 2 provides this comparison as an overview of differences between the methods and to consider each method's suitability for an organization.

This information can be used in planning improvements. As an example, the third method of unmanaged plain platform can plan to improve the low security resulting from its static long term secrets. The managed platform with Kubernetes might appear attractive given the high security, simpler trust components and low costs, but in terms of compatibility the platform is not a match. Instead the organization can look to the configuration management method to provide an improvement through dynamic short lived secrets at higher cost and complexity in trust, but no significant change to the platform.

Table 2: The five attributes of comparison for secure introduction

Attribute	**Managed platform with Kubernetes**	**Unmanaged platform with configuration management and HSM**	**Unmanaged plain platform**
Security	High	Medium	Low
Trust	Kubernetes infrastructure Kubernetes administrators	Configuration management software HSM card Configuration administrators System administrators	Operating system security Image management System administrators
Financial	None	Medium	None
Toil	Low	Medium	Low
Compatibility	Kubernetes platform High auditability Good support	Broad platforms Medium auditability Fast execution	Broad platforms Poor compliance Self-supported

6.2 Experiment demonstration

The scenarios in the previous subsection demonstrate differences between three methods of secure introduction. These ranged in risk, cost and technical details. Next, an experiment will simulate an attempt to derive an improved method and evaluate the results.

The third scenario was based on storing a credential directly on persistent storage. While at first glance this might appear attractive as it provides flexibility in terms of compatibility and financial investment, the level of security is low. A user with access to the host can view and duplicate the long lived credential. In a bid to improve on this situation a Trusted Platform Module (TPM) can be used. This is a device that is integrated in computers to manage private cryptographic keys and to perform some cryptographic operations. TPM is conceptually similar to HSM but provides fewer capabilities at much lower financial cost. TPM is most frequently used to support full disk encryption, but there are other use cases including some that do not yet exist in the marketplace (Arthur and Challener, 2015).

The scenario to be improved was based on an unmanaged platform without specialized hardware, but the addition of TPM does not necessarily constitute specialized hardware. A discrete TPM is included by default in some servers and today TPM functionality also exists in firmware (Raj et al., 2016)

The experiment will be described in terms of materials, setup, process and results to provide data to evaluate with the framework. Detailed experiment command inputs and outputs are provided in Appendix A.

The narrative of the experiment is as follows.

> *As a designer of methods of secure introduction I need to determine benefits and disadvantages of my work so that I can evaluate the results.*

6.2.1 Materials and setup

To prepare a client host for the experiment a basic Linux system was assembled using commodity components including CPU, motherboard, RAM, disks, power supply, and a case to house it all. The brand and features of these are not relevant except for ensuring that the motherboard model provided a TPM header. A TPM was not included but purchased separately from a retailer at low relative cost. To put the cost into perspective, the TPM was around the cost of a large pizza whereas an HSM costs two or three magnitudes more pizza. The free Linux-based operating system CentOS 8 was installed, selected for being recently released and providing a recent version of a TPM API ("TCG Software Stack (TSS) Specification," 2018). The secure introduction client software also consisted of the OpenSSL cryptography library and tools, the curl command-line transfer tool, and shell scripting to tie it all together.

A Raspberry Pi running Raspbian 10 served as the host for the centralized secrets store. The secret store software was the open source version of HashiCorp Vault in a minimal configuration with local file storage.

6.2.2 Experiment process

The first phase of the experiment was to configure the client host's TPM. The TSS tools were used to create a 2048-bit RSA public key paired with a private key residing in the TPM. Although this required some initial literature review to find the appropriate commands, they could easily be generalized and scripted for use at scale.

The second phase of the process was to configure the secure store. The secret store's JWT authentication method was configured with minimal options consisting only of a static list of public keys to trust; in this case only the key generated in the first step from the TPM. A role object to represent the client host was created in the secrets store and associated with permissions for reading secrets. This role restricts clients to be authenticated only if they match certain fields in the JWT; in this case the subject field was required to contain "client.example.com".

The third phase was to access the secrets store from the client as depicted in Figure 5. In step 1 a simple shell script on the client host assembled a JWT containing validity timestamps and the "client.example.com" subject. The script signed the JWT in steps 2 and 3 by invoking OpenSSL with the TSS engine to interact with the TPM, and then called the login endpoint on the secure store in step 4. The result was a session token for accessing the client's example secrets in step 5.

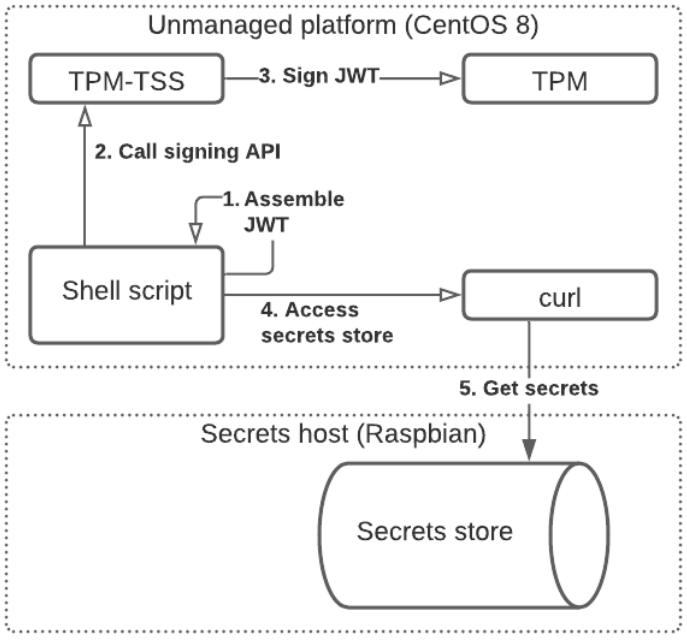

Figure 5: Accessing secrets with a TPM identity

6.2.3 Framework exercise

The TPM-based secure introduction experiment is intended to improve on the scenario where a credential was stored on persistent storage in a bare metal

server. The framework is used to determine whether it achieved this goal by evaluating the attributes and comparing them to the other scenario.

- Security: Medium
    - Confidentiality is protected by the TPM dynamically generating a signed credential for the host, and the token having a limited lifetime using a JWT expiration claim. The token was cryptographically authenticated.
    - Integrity and availability are protected by the closed system of the TPM. Tampering with the TPM could cause it to fail and disrupt access to secrets.
    - Security is improved over the bare metal scenario as a static credential on persistent storage can be recovered from the media. Being able to use a short duration and frequent regeneration of the token limits the risk if a token is intercepted
- Trust: TPM device, client software, operating system security, system administrator
    - The TPM signed the token on the client host which meant control of the TPM device implied the ability to sign arbitrary tokens.
    - The client script that generated the token was trusted to set claims to define the token scope but the client script could be modified to set a different subject or extended expiration time.
    - The operating system was protecting access to the TPM and the script.
    - A system administrator with super user privileges could circumvent the controls to sign new tokens using the TPM directly or by calling the client script.
    - This is not an improvement. The trust model becomes more complex with TPM compared to the static credential generated by an image creation tool. There are more distributed components to be trusted, and in some ways the client was trusted to not misrepresent itself, e.g. to set claims that expand privileges in a role based access control configuration.
- Financial: Low
    - CapEx is small as the TPM device presented a small cost compared to the cost of the rest of the hardware. There is no OpEx.
    - This is not an improvement, although the small cost was negligibly worse compared to the free cost of the static credential approach.
- Toil: Low
    - Low toil was required initially to develop the generalized

scripting for interacting with the TPM through the standardized TSS interface. However, mapping public keys to the permissions to access secrets may present some persistent toil for each bare metal server.

    - This is equivalent. After the public key infrastructure has been defined, managing the mappings of keys to permissions is similar to managing mappings of hosts to permissions in a centralized static provisioning scheme.

- Compatibility: Bare metal server platforms, compliant, complex support, fast execution
    - A bare metal server is required for the type of physical TPM used in this experiment. Installing dedicated hardware devices is not applicable to public cloud platforms, limiting this method to physical platforms with single tenants.
    - Compliance requirements are met as each server has a unique credential with limited duration and their private keys are never exposed as files.
    - Support is complex due to the addition of TPM devices. At a high level they are simple devices but their interfaces are made to be low level and flexible resulting in the need for detailed literature to understand how to initialize and operate the devices.
    - Execution speed is fast for on-going signing operations that can run asynchronously. The initial key generation was slow, taking several seconds, but this only occurs once or rarely.
    - This is not an improvement overall. Although the TPM is a standardized device, and can be included by default in commodity CPU architectures (Raj et al., 2016), the static credential approach is more versatile since it can be used on virtually any platform. Compliance is improved as each server has its own key and the credentials never need to be stored on disk. The support aspect is worsened as tools to configure and maintain this solution would need to be developed as a custom solution and maintained by the organization.

The conclusion of this experimental framework exercise is that the TPM-based method of secure introduction is only favorable depending on the objectives. The security is improved but the trust model becomes more complex. If organizational policy will not allow the required components to be trusted then the security improvements are negated. The compatibility was also reduced to limit this approach to be used on single-tenant bare metal hosts, and requires the organization to provide the support for the feature as it is not available from a third party.

Iterations to this method could improve the compatibility by automating the configuration and providing high level tools for preparing the TPM, provisioning the public key to the secrets store, and creating a versatile tool for

generating client certificates for authentication.

7 Evaluation

An evaluation of the designed artifact against the defined objectives is provided to round off the design science research process. The evaluation phase of the design science research methodology can "include any appropriate empirical evidence or logical proof" (Peffers et al., 2006). Two of their suggestions that are appropriate to this work include a comparison of the artifact against the solution objectives defined in the initial phase, as well as collecting feedback. The following evaluation consists of a first part discussing the results of the demonstration and a second part with expert feedback on the relevance of the five attributes of the framework.

7.1 Demonstration results

The main objective of the framework is to allow comparison of methods of secure introduction. In order to make comparisons some attributes need to be examined in each object, and determining which of those attributes are relevant is the core of this main objective. By studying literature and practice the three themes of risk, cost, and technical, were identified. These themes were then further broken down into the attributes of security, trust, financial, toil and compatibility. Whereas at first blush a comparison of secure introduction might suggest looking only at the purpose, which is to improve security, a look at the other four attributes gives a more complete picture. To be clear, analysis of the methods does reveal other attributes that differ but these were intentionally excluded or folded into an encompassing attribute, as they were generally less useful. For example, the performance of one method might differ from another, but since "introduction" implies a single occurrence per instance, the performance does not play a big role unless it is extraordinarily slow. Performance was therefore folded into compatibility because in a case where performance is a concern, a method that does not offer the performance would not be compatible with the requirements.

The objective was intended to be useful in at least two situations. Both of these were demonstrated by exercises in the Evaluation section; first in scenarios and second in an experiment.

7.1.1 Scenario evaluation

When a set of methods of secure introduction is presented there is not one single method that is the only appropriate choice for all situations. Wearing the hat of a system designer, making the best choice requires information specific to the organization about its tolerance for risk and costs on top of the details of the methods being considered. The framework artifact was demonstrated as being useful in helping to narrow down on distinct differences across both technical and social areas that would uncover gaps in security and trust risks, financial and operational costs, as well as platform

compatibility.

7.1.2 Experiment evaluation

As the experiment is simulates an attempt at improving a specific situation, namely that of the versatile but insecure method of distributing secrets by writing them directly into persistent files when the host is installed, the framework is exercised as one to one comparison. The experiment is described in terms of technique and cost and then compared to the original method attribute by attribute. This allowed the experimenter to determine how it stacked up, whether it is worthwhile in pursuing, and where to make further improvements.

7.2 Expert feedback

Feedback on the design was collected in order to round out the evaluation phase of the design science research. The feedback was collected from a set of respondents by survey.

7.2.1 Survey design

Expertise in enterprise IT security was desired of the respondents for this feedback. The pool of candidates was selected from a professional social network where ideally profiles reflected some experience in enterprise IT security or in system architecture and design which might imply some security aspects. All were employed or had been employed in that capacity in different enterprise software companies located either in the USA or Sweden. The survey included questions intended to determine some context. Multiple choice questions asked about the respondent's background in regard to their academic and professional experience with enterprise IT security in general, and with the theory, architecture and implementation of secure introduction specifically. A question was also included about experience with various platforms since it has been determined that secure introduction is very different depending on platform.

The survey started out with an introduction summarizing the concept of secure introduction of enterprise IT secrets and the three examples of Kubernetes, configuration manager and plain embedded secrets. It was important to set the stage because the specific term for secure introduction might differ even though the concept of secure introduction is well understood. The introduction ended by presenting a summary of the Design and development section presenting the five attributes of the framework and the reason for their inclusion. The main questions of the survey asked for feedback on the relevance of each attribute to be ranked on a scale of relevance: not relevant, barely relevant, somewhat relevant, or very relevant. The final question was a free form field for submitting anything deemed missing from the attributes.

7.2.2 Survey results

Seven people responded out of the ten who received the survey. Based on the background answers none of the respondents were filtered out as none reported lacking enterprise IT security familiarity and all had some familiarity with the theory, architecture and implementation of secure introduction. All respondents also had experience across multiple platforms.

The results of the question on framework attribute relevance are provided in Figure 6.

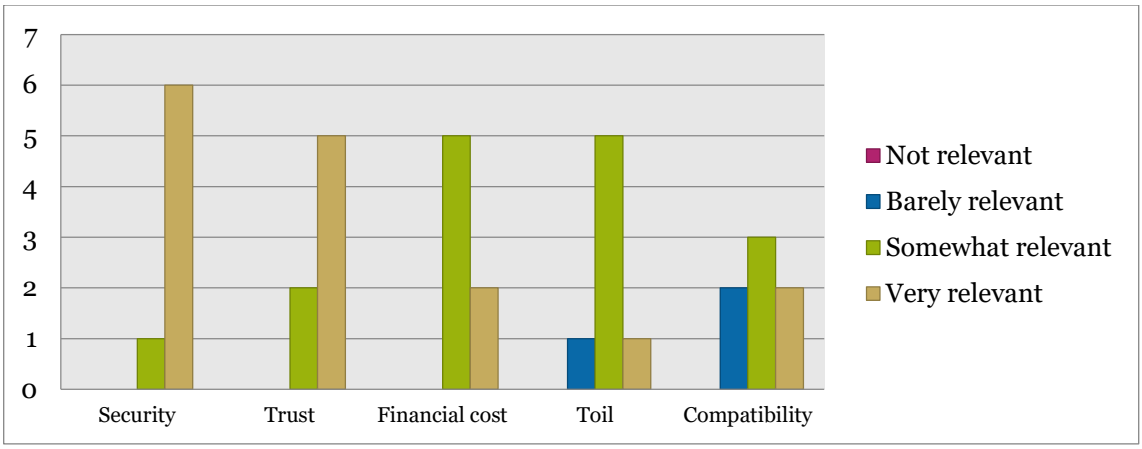

Figure 6: Attribute relevance survey results

The results for the first four attributes of security, trust, financial cost and toil are favorable in relevance, rating at least somewhat relevant by most respondents. Security and trust rated the highest, perhaps not surprisingly as these are likely the most associated with secure introduction no matter of their context. The attribute that proved to be the most divisive was that of compatibility. Due to the anonymous nature of the survey a clarification could not be asked, but the subsequent question on any missing attributes of the framework provides a clue.

Four areas were suggested as not being covered by the framework.

1. Ability to ensure controls are in place and auditable to follow security and regulatory standards.
    a. This could be stated in terms of features such as whether the method includes some integrated audit logging.
2. Long term support for the method.
    a. This was suggested twice. Support can be a concern, although perhaps more applicable to a particular vendor's feature than overall methods. In some cases they are one and the same though such as Kubernetes which is the platform and Kubernetes integrated service account is the basis of the method. In other cases the solution might need to be developed and self-supported by the organization.
3. Performance in terms of execution speed.

    a. The respondent suggests this might rarely be a concern unless exercised frequently. The term introduction implies a single occurrence so it likely is rarely a concern. In some implementations the start time might be impacted by lengthy introduction processes, or there could be a cycle for refreshing secrets.

4. Cloud service provider focus including secrets-as-a-service.
    a. This could be stated in terms of features; all components of the method are managed by the platform and infrastructure.

7.2.3 Iteration

After interpreting the survey feedback, the first four attributes of security, trust, financial cost and toil remain as initially defined. Only the last attribute, compatibility, is of concern. To address this concern the compatibility attribute was generalized as not being only specific to platform support. Instead it was broadened to allow the feedback on attributes missing from the framework to be integrated, as needed, into the compatibility attribute to improve its value.

Thus, in the first iteration of the framework the compatibility attribute was defined strictly as compatibility with the platform, i.e. bare metal server provides no controller-provided identities, so a method that depends on a controller cannot be used.

In the second iteration of the framework, which is subsequently reflected in the Design and development section, the compatibility attribute's description has been expanded to include compatibility with other requirements desired by the organization including compliance, support, speed of execution and hosting features.

8 Conclusion

Protecting secrets can be something of a rabbit hole as secrets can be protected using encryption, but the result is a new secret consisting of a decryption key. That decryption key can also be encrypted, but then there is yet another decryption key. At the bottom of this dependency chain we find some secret that is required in its verbatim form at the start of the process. Three different methods of introducing secrets to enterprise IT systems have been explored by this work with the goal of providing a method for evaluating the variety of methods available against the requirements of an organization. To that end a framework was designed to allow methods of secure introduction to be evaluated and compared. Adhering to a design science research method in six steps (Peffers et al., 2006), the problem was identified and objectives were defined. The framework was designed, resulting in five attributes to be examined in the different methods: security, trust, financial cost, toil and compatibility. The framework was demonstrated by running through three scenarios based on example methods, as well as in a practical example of an

exercise to improve a method of secure introduction. The work was evaluated by comparing the designed framework and demonstrations against the objectives, as well as by collecting feedback from experts. After an iteration and refinement the results are finally communicated through this paper.

The research question to be answered was how to compare methods of secure introduction, and the solution is the framework that has been designed to bridge the identified gap. The framework allows methods of secure introduction to be compared by evaluating five attributes that impact the method's success in an organization.

In recognition that one size does not fit all when it comes to enterprise security, the framework is intended to be high level to meet the diversity in enterprise organizations in terms of size, sector, and technology stacks and so on. However, future improvements to the framework could include specializations for some subset of use cases such as managed platforms. Going deeper, with a managed platform there are some root secrets that protect the secrets of the tenants. These secrets are beyond the scope of this work, but how are those root secrets ultimately managed, what lies at the bottom of the managed platform rabbit hole?

Another angle on future work could be in taxonomy. This work does not present a comprehensive review of methods of secure introduction, but this could be a useful resource for continuing academic research and application development in this space.

References

Amazon Elastic Compute Cloud [WWW Document], 2021. . Instance Identity Doc. URL https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-identity-documents.html (accessed 3.6.21).

Arthur, W., Challener, D., 2015. A practical guide to TPM 2.0: using the new Trusted Platform Module in the new age of security, The expert's voice in secure computing. Apress Open, New York, NY.

Beyer, B., Jones, C., Petoff, J., Murphy, N.R. (Eds.), 2016. Site reliability engineering: how Google runs production systems, First edition. ed. Oreilly, Beijing ; Boston.

Bradley, J., Sakimura, N., Jones, M.B., 2015. JSON Web Token (JWT) [WWW Document]. URL https://tools.ietf.org/html/rfc7519 (accessed 3.26.21).

Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., Polk, W., 2008. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile [WWW Document]. URL https://tools.ietf.org/html/rfc5280.html (accessed 4.24.21).

Fox, D., 2009. Hardware Security Module (HSM). Datenschutz Datensicherheit - DuD 33, 564–564. https://doi.org/10.1007/s11623-009-0145-9

Google Compute Engine [WWW Document], 2021. . Verif. Identity Instances. URL https://cloud.google.com/compute/docs/instances/verifying-instance-identity (accessed 3.6.21).

Hevner, A., Chatterjee, S., 2010. Design Research in Information Systems, Integrated Series in Information Systems. Springer US, Boston, MA. https://doi.org/10.1007/978-1-4419-5653-8

Humble, J., Farley, D., 2010. Continuous delivery: reliable software releases through build, test, and deployment automation. Addison-Wesley, Upper Saddle River, NJ.

Initiative, J.T.F.T., 2012. Guide for Conducting Risk Assessments (No. NIST Special Publication (SP) 800-30 Rev. 1). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-30r1

International Organization for Standardization, 2018. ISO/IEC 27005:2018 [WWW Document]. ISO. URL https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/07/52/75281.html (accessed 3.13.21).

International Organization for Standardization, 2013. ISO/IEC 27001:2013 [WWW Document]. ISO. URL https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/05/45/54534.html (accessed 3.13.21).

Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H., Carriere, J., 1998. The architecture tradeoff analysis method, in: Proceedings. Fourth IEEE International Conference on Engineering of Complex Computer Systems (Cat. No.98EX193). Presented at the Proceedings. Fourth IEEE International Conference on Engineering of Complex Computer Systems (Cat. No.98EX193), pp. 68–78. https://doi.org/10.1109/ICECCS.1998.706657

Kindervag, J., Balaouras, S., Mak, K., Blackborow, J., 2016. No More Chewy Centers: The Zero Trust Model Of Information Security [WWW Document]. URL https://www.forrester.com/report/No+More+Chewy+Centers+The+Zero+Trust+Model+Of+Information+Security/-/E-RES56682 (accessed 4.21.21).

Kubernetes Authentication [WWW Document], 2021. . Authenticating. URL https://kubernetes.io/docs/reference/access-authn-authz/authentication/ (accessed 3.6.21).

Kubernetes Components [WWW Document], 2021. . Kubernetes. URL https://kubernetes.io/docs/concepts/overview/components/ (accessed 3.26.21).

Kubernetes Version Skew Policy [WWW Document], 2021. . Kubernetes. URL https://kubernetes.io/releases/version-skew-policy/ (accessed 6.1.21).

Matej Artač, Tadej Borovšak, Elisabetta Di Nitto, Michele Guerriero, Damian Andrew Tamburri, 2017. DevOps : introducing infrastructure-as-code. Softw. Eng. Companion, International Conference on Software Engineering 497–498. https://doi.org/10.1109/ICSE-C.2017.162

National Institute of Standards and Technology, 2020. Security and Privacy Controls for Information Systems and Organizations (No. NIST Special Publication (SP) 800-53 Rev. 5). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-53r5

NGINX [WWW Document], 2021. . ngx_http_ssl_module. URL http://nginx.org/en/docs/http/ngx_http_ssl_module.html#ssl_password_file (accessed 4.15.21).

Palmius, J., 2007. Criteria for Measuring and Comparing Information systems. 30th Inf. Syst. Res. Semin. Scand. IRIS-30.

Peffers, K., Tuunanen, T., Gengler, C., Rossi, M., Hui, W., Virtanen, V., Bragge, J., 2006. The design science research process: A model for producing and presenting information systems research. Proc. First Int. Conf. Des. Sci. Res. Inf. Syst. Technol. DESRIST.

Perrin, C., 2008. The CIA Triad [WWW Document]. TechRepublic. URL https://www.techrepublic.com/blog/it-security/the-cia-triad/ (accessed 3.13.21).

Rahman, A., Partho, A., Morrison, P., Williams, L., 2018. What Questions Do Programmers Ask about Configuration as Code? 2018 IEEEACM 4th Int. Workshop Rapid Contin. Softw. Eng. RCoSE Rapid Contin. Softw. Eng. RCoSE 2018 IEEEACM 4th Int. Workshop RCOSE 16–22.

Raj, H., Saroiu, S., Wolman, A., Aigner, R., Cox, J., England, P., Fenner, C., Kinshumann, K., Loeser, J., Mattoon, D., Nystrom, M., Robinson, D., Spiger, R., Thom, S., Wooten, D., 2016. fTPM: A Software-Only Implementation of a TPM Chip. Presented at the 25th USENIX Security Symposium (USENIX Security 16), pp. 841–856.

Rose, S., Borchert, O., Mitchell, S., Connelly, S., 2020. Zero Trust Architecture (No. NIST Special Publication (SP) 800-207). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-207

Saltzer, J.H., 1974. Protection and the control of information sharing in multics.

Commun. ACM 17, 388–402. https://doi.org/10.1145/361011.361067

Stoermer, C., Bachmann, F., Verhoef, C., 2003. SACAM: The Software Architecture Comparison Analysis Method. CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST.

TCG Software Stack (TSS) Specification, 2018. . Trust. Comput. Group. URL https://trustedcomputinggroup.org/resource/tcg-software-stack-tss-specification/ (accessed 4.5.21).

Vault Auth Methods [WWW Document], 2021. . Vault HashiCorp. URL https://www.vaultproject.io/docs/auth (accessed 3.13.21).

Yeoman, S., 2019. How secure are bare metal servers? Netw. Secur. 2019, 16–17. https://doi.org/10.1016/S1353-4858(19)30024-8

Appendix A

Materials

Client

- curl 7.61.1
- OpenSSL 1.1.1k
- TPM 2.0 NUVOTON NPCT650
- tpm2-tools 4.1.1
- tpm2-tss 2.3.2
- tpm2-tss-engine 1.1.0

Secrets store

- HashiCorp Vault 1.7.1

Client host configuration

Create the RSA signing key in the TPM, saving the wrapped key as the file private.tss

```
$ tpm2tss-genkey -a rsa -s 2048 private.tss
```

Export the associated RSA public key, saving as the file public.pem

```
$ openssl rsa -engine tpm2tss -inform engine -in private.tss -pubout -outform pem \
  -out public.pem
engine "tpm2tss" set.
writing RSA key
```

Prepare script for creating a JWT and signing using the RSA private key in the TPM

```
$ cat create_jwt.sh
#!/bin/bash

sub="$1"

# Find the issued-at and expiration epoch times
iat="$(date +%s)"
exp="$((timenow + 3600))"

# Simple JWT structure
header="{\"typ\":\"JWT\",\"alg\":\"RS256\"}"
claims="{\"sub\":\"${sub}\",\"iat\":${iat},\"exp\":${exp}}"

# Encode the input in URL-safe base64
enc() {
  echo -n $(base64 -w0 <<< $1 | tr -d '=\n' | tr '/+' '_-')
}

# Create a signature of the input using a key in the TPM
sign() {
  echo -n $1 | openssl dgst -sha256 -engine tpm2tss -keyform engine \
    -sign private.tss -binary | base64 -w0 | tr '+/=' '-_ '
}

jwt_enc=$(enc "${header}").$(enc "${claims}")
sig_enc=$(sign ${jwt_enc})
```

```
echo ${jwt_enc}.${sig_enc}
```

Secrets store configuration

Create an example secret and an access policy for listing and reading the secret

```
$ vault secrets enable -path=secrets -version=1 kv
Success! Enabled the kv secrets engine at: secrets/

$ vault write secrets/example/password value=swordfish
Success! Data written to: secrets/example/password

$ vault policy write example-policy - << EOF
> path "secrets/example/*" {
>   capabilities = ["list", "read"]
> }
> EOF
Success! Uploaded policy: example-policy
```

Configure JWT authentication, trusting the TPM RSA public key

```
$ vault auth enable -path=example/jwt jwt
Success! Enabled jwt auth method at: example/jwt/

$ vault write auth/example/jwt/config jwt_validation_pubkeys=@public.pem
Success! Data written to: auth/example/jwt/config
```

Create a JWT authentication role, attaching the policy authorizing access to the secret

```
$ vault write auth/example/jwt/role/example role_type=jwt \
  bound_subject=client.example.com user_claim=sub policies=example-policy
```

Secrets access

Create a signed JWT using the script prepared on the client host, saving the JWT to an environment variable

```
$ jwt=$(./create_jwt.sh client.example.com)
engine "tpm2tss" set.

$ echo $jwt
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9Cg.eyJzdWIiOiJjbGllbnQuZXhhbXBsZS5jb20iLCJpYXQiOjE2M
jM1NDU5NTUsImV4cCI6MTYyMzU0OTU1NX0K.SuWdx1u144q-_0eaovffGqzhQT9wHWbu55TmMIg8AmFSurcAf4gc
PMoNpsNU-22VzQax9zf5SGD-oCJ_Elp8Ab8SNhnMI06HYbUIVFg89IsPdctBdmYjx2wwGLkHcHWoNIDXe45GCKmF
upOjuj2OMFCcqSfVetSvJpJvA8QUe6XjblWKlE6c79dhmTTe3Ts0GdOshGFldrE5OfhUL9Af3saYUtG50yDTGp2x
XkSfvQMRDcZNLGT5WuopqUDGrLVec6BEAQCCVse4aePDEgekzYKulvI9UNk10t1vtj1NbMgxFaTCfFQnQjLP4Mo8
rBfe4gOsKECNIfHLBxk_I935Ng
```

Authenticate against secrets store JWT login API to get a Vault access token

```
$ curl -s -X POST -d "{\"jwt\": \"${jwt}\", \"role\": \"example\"}" \
  $VAULT_ADDR/v1/auth/example/jwt/login | jq .auth.client_token,.auth.token_policies
"s.spWSVfrxKJhZZtXNNq7BhbmR"
[
  "default",
  "example-policy"
]
```

List and read the secret using the Vault access token

```
$ curl -s -X LIST -H "X-Vault-Token: s.spWSVfrxKJhZZtXNNq7BhbmR" \
>   $VAULT_ADDR/v1/secrets/example | jq .data
{
  "keys": [
    "password"
  ]
}

$ curl -s -X GET -H "X-Vault-Token: s.spWSVfrxKJhZZtXNNq7BhbmR" \
>   $VAULT_ADDR/v1/secrets/example/password | jq .data
{
  "value": "swordfish"
}
```