**УДК 62** **Технические науки**

***Самородских Илья Леонидович,*** *бакалавр, Московский Государственный Технический Университет им Н.Э. Баумана, Россия, г. Москва*

# СИСТЕМЫ УПРАВЛЕНИЕ СЕКРЕТАМИ

**Аннотация:** В настоящей статье была рассмотрена проблема безопасности серверов, исследовано понятие систем управления секретами и где они применяются, произведено сравнение существующих технологий.

**Ключевые слова:** секреты, аутентификация, авторизация.

**Annotation:** This article discusses the problem of server security, examines the conce pt of secret managemen t systems and where they are used, and compares existing technologies.

**Keywords:** secrets, authentication, authorization.

Системы управления секретами делают именно то, что подразумевает название: они хранят, управляют и предоставляют секреты. Эта технология решает проблемы, о которых большинство специалистов по безопасности еще не знает. По мере того, как команды разработчиков используют методы автоматизации и оркестровки, они создают проблемы безопасности, которые необходимо решать. Автоматизация облачных сервисов делает приложения и IT-сервисы более быстрыми и устойчивыми, но с этим улучшением возникает необходимость предоставления разрешений компьютерам и программному обеспечению, а не просто людям. В некотором смысле это новая итерация проблемы управления идентификацией с начала 2000-х годов, которая усилилась в отношении персональных устройств и инфраструктуры облачных вычислений.

И это является источником проблемы: разработчики автоматизируют сборку и тестирование ПО, а системные администраторы автоматизирует предоставление ресурсов, но оба лагеря по-прежнему считают, что безопасность замедляет их. Практики непрерывной интеграции и DevOps повышают гибкость, но также создают риски для безопасности, включая хранение секретов в репозиториях исходного кода и хранение учетных данных. Эта вредная привычка оставляет все части ПО, которые идут в производство, в опасности.

Все ПО нуждается в учетных данных для доступа к другим ресурсам: взаимодействовать с базами данных, получать ключи шифрования и получать доступ к другим услугам. Но эти привилегии доступа должны быть тщательно защищены, чтобы они не были использованы злоумышленниками или третьими лицами. Проблема заключается в том, что необходимо знать, какие права необходимо предоставлять, в каком формате ПО может принимать их, а затем безопасно предоставлять права доступа, когда человек не вовлечен или не может быть непосредственно вовлечен. Разработчики работают с источниками для идентификации, такими как Active Directory, но, как правило, неосведомленны о технологиях, которые помогают им распределять учетные данные по назначению.

Чтобы учесть эти изменения, были разработаны мощные утилиты и платформы для защиты конфиденциальных данных и безопасного распределения привилегий. Термин для этого нового класса продукта - «Управление Секретами»; и он меняет то, как мы предоставляем учетные записи, секреты и токены. Фактически, Управление Секретами - это ключевой элемент для перемещения DevOps в SecDevOps. В этом исследовании будет рассмотрено, почему это проблема для многих организаций, какие проблемы решают эти новые платформы и как они работают в новых средах.

**Проблема аутентификации и авторизации серверов**

Получение секретов крайне важно для функционирования алгоритмов автоматизации, но многие организации придерживаются классического

(простого) режима работы: секреты помещаются в файлы или встраиваются в алгоритмы, чтобы задачи могли выполняться без вмешательства человека. Разработчики понимают, что это является проблемой безопасности, но продолжают работать в таком режиме и не рассказывают службе безопасности о том, как они предоставляют секреты. Поэтому большинство архитекторов безопасности не знают об этой возникающей проблеме, пока не произойдет соответсвующее событие [1].

Проблема не нова. Системные администраторы хранили секреты в незащищенных файлах в течение десятилетий. Ни один администратор не хочет, чтобы посреди ночи вызывали на работу, чтобы ввести пароль, чтобы приложение могло перезапуститься. ИТ-администраторы обычно хранят ключи шифрования в файлах, чтобы операционная система или ПО могли обращаться к ним при необходимости. Администраторы базы данных помещали ключи шифрования и пароли в файлы для облегчения автоматической перезагрузки до тех пор, пока корпоративные сети не начали подвергаться большим атакам, и эта отраслевая практика была запрещена. С тех пор специалисты использовали многое: от ручного ввода до серверов управления ключами и даже аппаратных ключей. Но старые подходы не обеспечивают необходимой безопасности в новых вычислительных средах и средах разработки - и ставки намного выше из-за динамического характера того, как предоставляются программное обеспечение и услуги.

Помимо преимуществ, автоматизация и оркестровка порождают новые проблемы безопасности. Основной проблемой сегодня является надежное распространение секретной информации. Это особенно проблематично, когда «машины» - это не нечто, стоящее в стойке на объекте совместного размещения, а вместо этого эфемерный экземпляр машины - или, возможно, тысячи экземпляров, работающих одновременно. Как можно отследить, какой сервер, виртуальная машина или контейнер чем является, или какой набор разрешений предоставить каждому? Масштабы проблемы расширяются по мере того, как расширяются возможности автоматизации между командами.

Команды разработчиков должны обмениваться данными, конфигурациями и ключами доступа между собой для совместной работы по разработке и тестированию приложений. Автоматизированные серверы сборки нуждаются в доступе к управлению исходным кодом, шлюзами API и ролям пользователей для выполнения своих задач. Серверы должны иметь доступ к зашифрованным дискам, приложениям - к базам данных, а контейнерам должны быть предоставлены привилегии при запуске. Автоматизированные сервисы не могут ждать, пока пользователи введут пароли или предоставят учетные данные поэтому нужны гибкие и автоматизированные методы для предоставления данных, удостоверений и прав доступа [2].

**Случаи применения систем управления секретами**

Существует несколько причин, по которым необходимо управление секретами, а варианты использования разнообразны. Тем не менее, ключевым вопросом для решения является безопасное предоставление прав доступа к сервисам, которое в настоящее время практически отсутствуют.

**Шлюзы API и ключи доступа**

Интерфейсы прикладного программирования (API) - это средства взаимодействия программ с другим программным обеспечением и службами. Эти API формируют базовый интерфейс для скоординированной работы. Чтобы использовать API, надо сначала пройти аутентификацию своей личности или своего кода в шлюзе API. Обычно это выполняется путем предоставления ключа доступа, токена или ответа на криптографический запрос. Для простоты автоматизации многие разработчики вставляют в код ключи доступа, оставляя себя уязвимыми для простого просмотра файлов или кода. И слишком часто, даже если они хранятся в личном файле на рабочем столе разработчика, ключи случайно передаются или публикуются - иногда в общедоступных хранилищах кода. Цель здесь - сохранить ключи доступа в секрете, в то же время предоставляя их действующим приложениям по мере необходимости [3].

**Сервисы**

Приложения редко являются самостоятельными объектами. Обычно они состоят из множества различных компонентов, баз данных и вспомогательных служб. В современных архитектурах приложений запускается множество экземпляров приложения, чтобы обеспечить масштабируемость и отказоустойчивость. Когда приложения запускаются, будь то в контейнерах или поверх виртуальных машин, или серверов, мы должны предоставлять им данные конфигурации, сертификаты идентификации и токены. В связи с наличием большого количества экземпляров возникают вопросы:

– как недавно созданная виртуальная машина, контейнер или приложение обнаруживают свою личность и получают доступ к необходимым ресурсам;

– как можно однозначно идентифицировать конкретный экземпляр контейнера среди остальных клонов.

В погоне за полной автоматизацией своей среды организации автоматизировали свою работу так быстро, что, как правило, результат получается с минимальной безопасностью и перевесом в сторону скорости сборки. Разработчики обычно помещают учетные данные в конфигурационные файлы, которые удобно доступны для приложений и серверов при запуске. Они также часто используются совместно с другими приложениями и службами, которые не должны иметь доступа. Цель состоит в том, чтобы разделить полномочия, не вызывая поломок или неприемлемых барьеров.

**Автоматизирование сборки ПО**

Большинство сред сборок ПО небезопасны. Разработчики считают, что безопасность в процессе разработки замедляет их, поэтому они часто обходят контроль безопасности в процессах разработки [4]. Среды сборки, как правило, находятся под контролем разработчиков на серверах, принадлежащих разработчикам, поэтому мало кто знает, где они находятся и как они работают. Серверы для ночных сборок существуют уже более десяти лет, причем постоянно повышается автоматизация для повышения гибкости. Поскольку весь данный процесс ускоряется, человеческий контроль в большей степени

отдаляется. По мере того, как новый код, шаблоны формирования и сценарии регистрируются в репозиториях, серверы сборки, такие как Jenkins, автоматически восстанавливают приложения. Но они также проверяют новые дополнения, выполняющие проверки качества и безопасности, останавливая процесс в случае неудачи этих тестов. Это означает, что качественные тесты и тесты безопасности являются частью процесса сборки и больше не замедляют процесс разработки, а скорее служат защитой, чтобы плохой код больше не попал в релиз. Ввод секретов в процесс, поскольку он также является частью процесса автоматизации, такой же динамичный и автоматический, как создание и запуск приложений. С помощью политик определяются права, которые должны быть предоставлены, гарантируя, что код и услуги предоставлены безопасно и без необходимости вмешательства человека [5].

**Предоставление идентификационных данных серверам**

Поскольку сейчас используются облачные сервисы, определение слово “сервер” меняется. Облако использует концепции вычислений, сети и хранилища как сервисов, которые распределяются по мере необходимости и удаляются, когда ненужны. Таким образом, то, что ранее рассматривалось как “машины”, стало эфемерной сущностью машины, часто являющейся виртуальной; представлен в виде образа сервера, контейнера или какой-либо аналогичной концепции. Но так как эти “машины” часто запускаются и отключаются, стало очень трудно отслеживать, какая из них ответила на какой запрос, поэтому нужен способ выдачи им уникальных идентификаторов. В некоторый момент может потребоваться найти потенциально сломанный или скомпрометированный экземпляр и выполнить реагирование на инцидент, но нельзя выполнять одно и то же корректирующее действие в отношении каждого экземпляра, поэтому уникальная идентификация имеет решающее значение.

**Шифрование данных**

Предоставление ключей шифрования для разблокировки зашифрованных томов и файловых хранилищ является обычной задачей как для локальных, так

и для облачных служб. Традиционно используются серверы управления ключами, предназначенные для безопасного распределения и управления ключами, но ряд коммерческих инструментов управления ключами (как аппаратных, так и программных) еще не был расширен для инфраструктуры или платформы как сервиса. Кроме того, разработчики требуют лучшей интеграции API для бесшовного использования с приложениями. Этой возможности часто не хватает, поэтому некоторые команды используют облачное управление ключами, в то время как другие выбирают системы управление секретами в качестве замены.

**Обмен секретами**

Программное обеспечение для совместной работы помогло командам разработки, обеспечения качества и управления продуктами сотрудничать в проектах, даже если люди работают удаленно или в компании, где команды находятся на разных континентах. В некоторых случаях возникает проблема безопасного обмена информацией между командой удаленных разработчиков или проблема передачи секретных данных между несколькими центрами обработки данных, не раскрывая их в явном виде. Базы данных, в которых хранятся данные для служб чата и совместной работы, как правило, не очень безопасны. Решение представляет собой надежный центральный репозиторий, где выбранная группа пользователей может хранить и получать секреты.

**Обзор технологий систем управления секретами**

Поскольку системы управления секретами - это новый класс продуктов, на рынке практически отсутствует единообразие характеристик. Многие из систем называются «продуктами», но являются всего лишь инструментами для личного пользования. Полнофункциональные системы корпоративного класса являются исключением. Однако существуют базовые функции, которые должна решать каждая система управления секретами:

– безопасное хранение секретов;

– передача секретов;

– управление идентификацией;

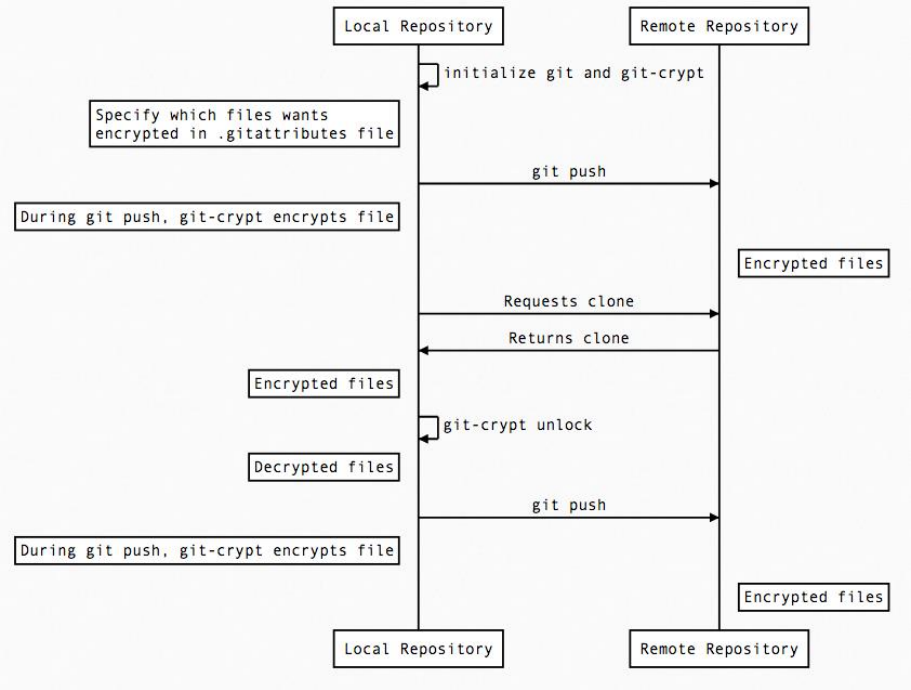
– доступ к управлению посредством API, а также зачастую посредством командной строки и веб интерфейса.

Следует иметь в виду, что большинство из них изначально разрабатывалось для выполнения одной очень конкретной задачи, например, внедрение секретов в контейнеры “на ходу”, или тесная интеграция с сервером сборки Jenkins, или дополнение облачный сервиса идентификации. Они хорошо выполняют одну задачу, но редко все сразу.

Далее рассмотрим два вспомогательных сервиса Git-crypt и AWS Система Управления Ключами, затем несколько полнофункциональных систем управления секретами и после этого перейдем к обобщенному сравнению самых популярных систем на рынке [6; 7].

**Git-crypt**

В начале рассмотрим систему, которую правильно использовать в любой компании, где применяются git репозитории. Git-crypt (рис. 1) - это проект с открытым исходным кодом для шифрования файлов в репозитории git. Опираясь на перехватчики git, файлы, перечисленные в файле. gitattributes, шифруются до того, как репозиторий передается на удаленный компьютер [8]. Для дешифрования необходимо клонировать репозиторий и выполнить команду git-crypt, указав путь к ключу, который использовался для



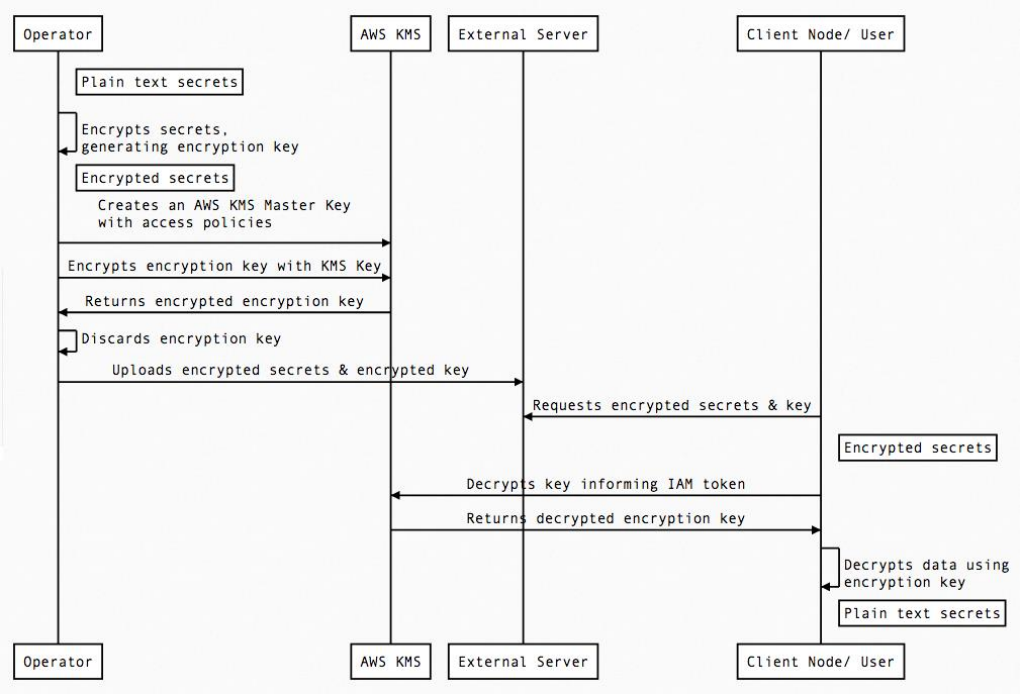

шифрования. Диаграмма последовательности, описывающая этот рабочий процесс, приведена ниже.

Рисунок 1. Диаграмма последовательности git-crypt

Особенность git-crypt - возможность указать более одного ключа для репозитория. Это означает, что секреты боевых репозиториев и репозиториев разработчиков могут храниться в одном месте, но только пользователи, имеющие доступ к правильным ключам, смогут расшифровать секреты.

**AWS Система Управления Ключами**

Вместо хранения секретов можно использовать Систему Управления Ключами AWS (рис. 2) для шифрования ключа шифрования. Если есть необходимость зашифровать версия оригинального ключа шифрования, то после этого его нельзя использовать для расшифровки ваших секретов. Поэтому безопасно хранить зашифрованную версию вместе с другими секретами. После этого можно оставить (или сохранить в надежном резервном хранилище) исходный ключ шифрования и доверить AWS расшифровку зашифрованного ключа шифрования. Диаграмма последовательности,




описывающая этот рабочий процесс, приведена ниже.

Рисунок 2. Диаграмма последовательности Системы Управления Ключами AWS

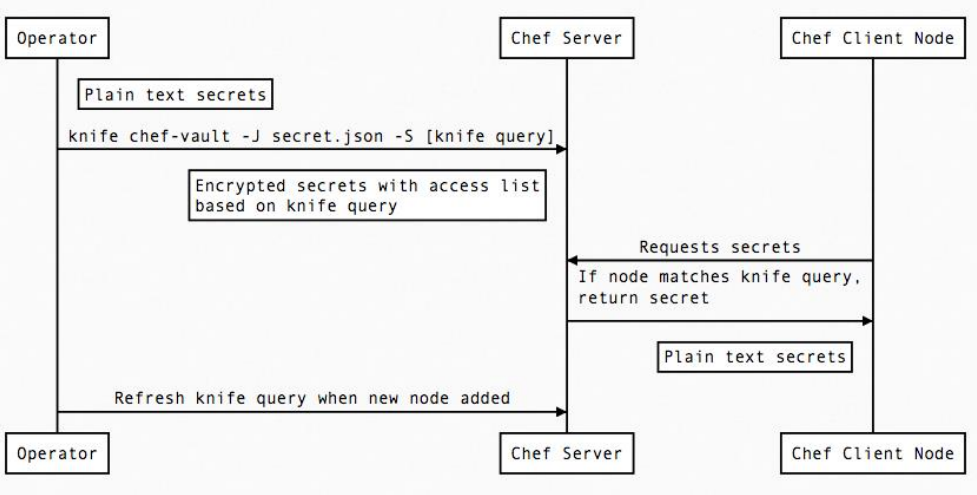
Одним из недостатков этого решения является зависимость от доступности сервиса AWS. А очевидным преимуществом этого решения является низкие сложность и связанные с этим расходы, которые могут послужить основой для простой стратегии управления секретами.

**Chef Vault**

Chef (рис. 3) - это инструмент, который позволяет описать, как должен выглядеть экземпляр сервера: какие пользователи должны быть зарегистрированы, какие приложения установлены, какие порты открыты и т. д. Эти описания хранятся в «рецептах» (recipes), которые содержатся в «кулинарных книгах» (cookbooks).

Chef Vault является дополнением к этому инструменту, позволяя клиентам-узлам Chef получать секреты от сервера Chef [9].

Часто могут потребоваться секреты, чтобы выполнить ряд задач в процессе инициализации. Например, могут потребоваться пароли для установки сервера базы данных или создания новых пользователей системы. В этих ситуациях




сервер (клиентский узел Chef) будет следовать следующему рабочему процессу:

Рисунок 3. Диаграмма последовательности Chef

**Conjur**

Conjur - это приложение с закрытым исходным кодом, которое выполняет управление секретами и общее управление каталогами и доступом с моделью RBAC. Приложение является автономным и предоставляется в виде образа Docker или AWS AMI. Предоставляются интерфейсы UI и CLI к ядру REST

API. В качестве каталога Conjur также предоставляет конечную точку LDAP для интеграции с другими приложениями, использующими каталог. Для секретов Conjur предлагает плагин Summon для предоставления секретов в качестве переменных среды.

Реализована система с использованием Ruby, а официальная документация сервера перечисляет такие службы, как Postgres и Nginx, в контейнере. Можно запустить несколько экземпляров приложения в виде «мастер-слейв».

**Keywhiz**

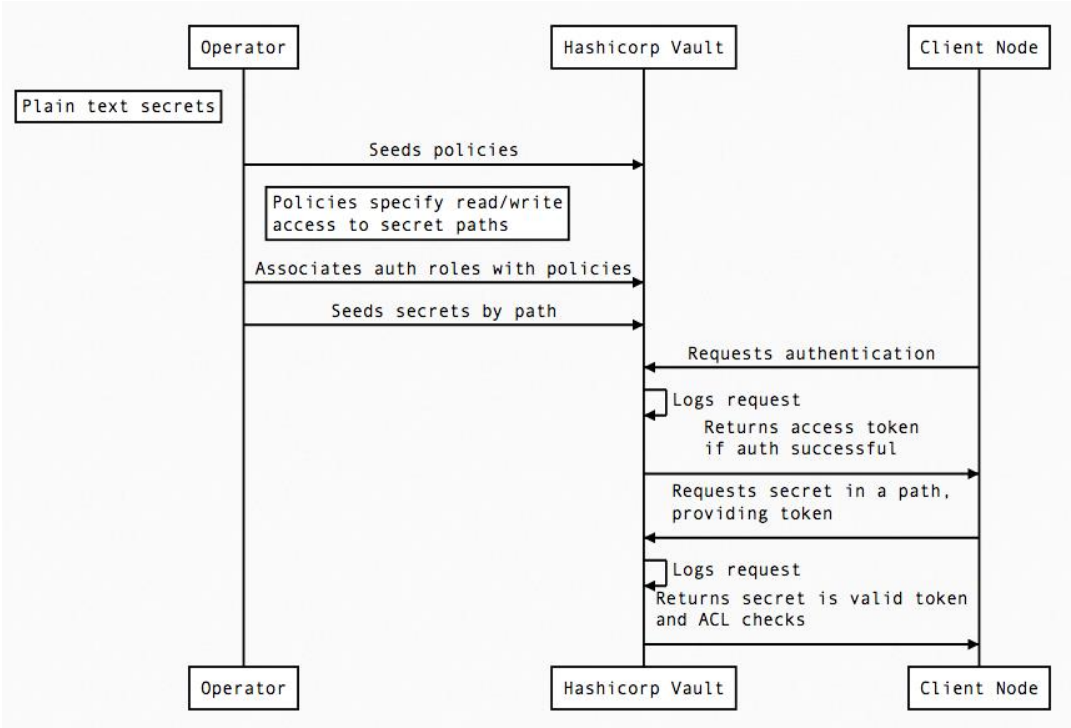
Keywhiz помогает распространять секреты инфраструктуры для сервисов. Это Java-сервис с JSON API и поддерживаемыми хранилищами: MySQL или Postgres. Клиентское приложение предоставляет модуль FUSE для секретов. Аутентификация выполняется с использованием взаимного TLS сертификата с использованием клиентского сертификата, поэтому для использования Keywhiz необходим PKI, который может предоставлять сертификаты службам и людям. Доступ открыт через командную строку, а также через веб-интерфейс для удобного управления секретами.

Есть некоторые ограничения в работе приложения, например, секреты имеют глобально уникальное имя, а группы не могут наследоваться от других групп. Square считают, что код на уровне альфа версии, но сервис полноценно используется внутри компании. Несмотря на то, что система появилась совсем недавно, список конечных точек API уж содержит несколько версий похожих функций, что может сбить с толку пользователей [10].

**Hashicorp Vault**

Самая многофункциональная, активно разрабатывающаяся более 3 лет и засчет этого наиболее популярная система, которая предоставляет хранилище секретов с возможностями управления. Секреты расположены в дереве с использованием ACL политик, ограничивающими доступ и разрешенные операции. Система позволяет контролировать доступ пользователем точечно к каждому секрету и каждой настройке, позволяет вести детальные логи и TTL

(время жизни) доступа к секретам. Vault поддерживает различные механизмы




аутентификации и бэкэнды секретов, что позволяет осуществить развертывание с высокой степенью доступности.

Рабочий процесс, показывающий хранение и получение секретов из Hashicorp Vault, изображен на рисунке 4.

Рисунок 4. Диаграмма последовательности Hashicorp Vault

Для некоторых конкретных служб Vault предлагает генераторы паролей, в которых новый секрет может быть создан для каждого запроса и отозван по истечении срока его действия. Официальные нструменты разработчиков для Vault включают клиент CLI, веб интерфейс, API интерфейс и библиотеки Ruby и Go. Также растет число сторонних инструментов и библиотек. Исходный код приложения находится в открытом доступе в официальном репозитории компании Hashicorp на GitHub.

Очевидным недостатком использования Hashicorp Vault является зачастую излишняя сложность, которую он представляет. Даже несмотря на

простую настройку, он все же требует, чтобы отдельный член команды отвечал за поддержание и управление списком доступа и самим сервером Vault.

**Сравнение функций самых популярных систем на рынке**

В таблице 1 представлено сравнение наиболее популярных систем управления секретами и вспомогательных систем на основе наличия того или иного функционала. Таблица составлена на основе найденной информации в официальной документации каждой системы, поэтому, если функция не заявлена и не описана, то считается, что она не представлена в системе.

| | **Генератор секретов** | **Управление версиями** | **Логирование** | **Высокая доступность** | **Интерфейсы** | **Языковая библиотека** |
|---|---|---|---|---|---|---|
| **Chef Vault** | Нет | Нет | Нет | Нет | CLI | Ruby |
| **Citadel** | Нет | Возможна с S3 | Возможна с S3 | Да | AWS | Ruby |
| **Conjur** | Требуется плагин | Да | Да | Да | UI, CLI | Ruby, Python |
| **Hashicorp Vault** | Да | Да | Да | Да | UI, CLI | Go, Ruby |
| **Keywhiz** | Требуется плагин | Да | Да | Нет | UI, CLI | Java |
| **Ansible Vault** | Нет | Через git | Нет | Да | CLI | - |
| **AWS KMS** | Нет | Нет | Нет | Нет | CLI | - |
| **Git-crypt** | Нет | Нет | Нет | Нет | CLI | - |

Таблица 1. Сравнение основного функционала систем управления секретами

**Библиографический список:**

1. What are the best shared secret managers? [Электронный ресурс] // Slant URL: https://www.slant.co/topics/2448/~best-shared-secret-managers (Дата обращения 02.06.2019).

2. What are the best Secrets Management Tools? [Электронный ресурс] // Stackshare URL: https://stackshare.io/secrets-management (Дата обращения 02.06.2019).

3. Hashicorp Vault [Электронный ресурс] // Hashicorp URL: https://www.hashicorp.com/products/vault/ (Дата обращения 02.06.2019).

4. Git-Crypt [Электронный ресурс] // GitHub URL: https://github.com/AGWA/git-crypt (Дата обращения 02.06.2019).

5. AWS Key Management Service [Электронный ресурс] // AWS URL: https://aws.amazon.com/ru/kms/features/ (Дата обращения 02.06.2019).

6. Ansible Vault [Электронный ресурс] // Ansible Documentation URL: https://docs.ansible.com/ansible/latest/user_guide/vault.html (Дата обращения 02.06.2019).

7. Chef-vault [Электронный ресурс] // Learn Chef URL: https://docs.chef.io/chef_vault.html (Дата обращения 02.06.2019).

8. Conjur Docs [Электронный ресурс] // Ceberark Conjur URL: https://docs.conjur.org/latest/en/Content/Resources/_TopNav/cc_Home.htm (Дата обращения 02.06.2019).

9. Keywhiz [Электронный ресурс] // Keywhiz URL: https://square.github.io/keywhiz/ (Дата обращения 02.06.2019).

10. Citadel [Электронный ресурс] // GitHub URL: https://github.com/poise/citadel (Дата обращения 02.06.2019).