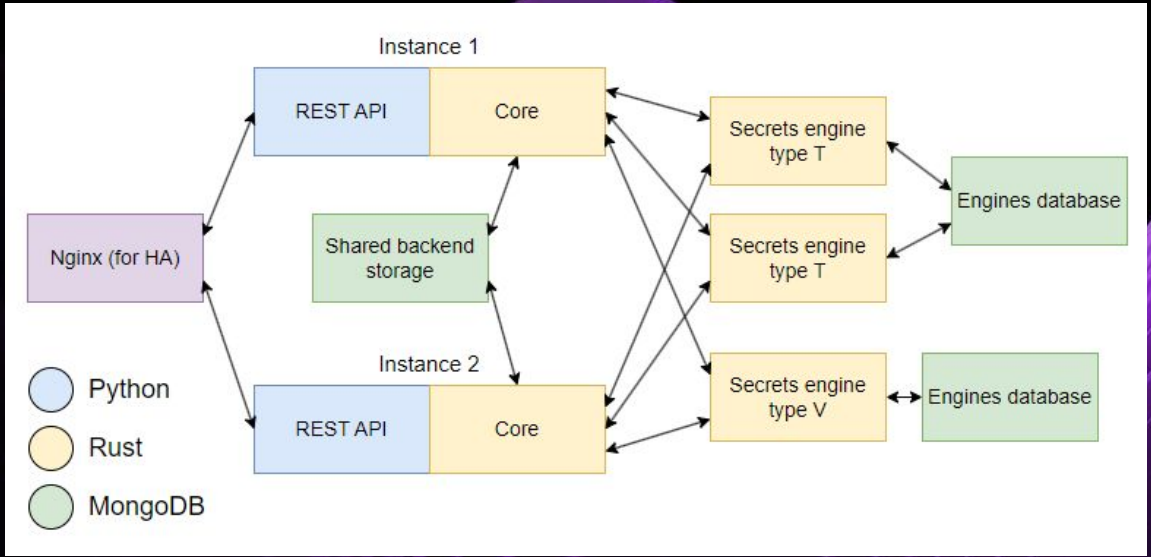
MORE.
Tech

reVault

Hashicorp vault переписанный на Rust и Python с идеями лучшего аудита

Команда ProSecCyberMicroLabsBusinessVupsenPupsenExprAICorpAGIFactoryTechSoftTeam Inc ltd

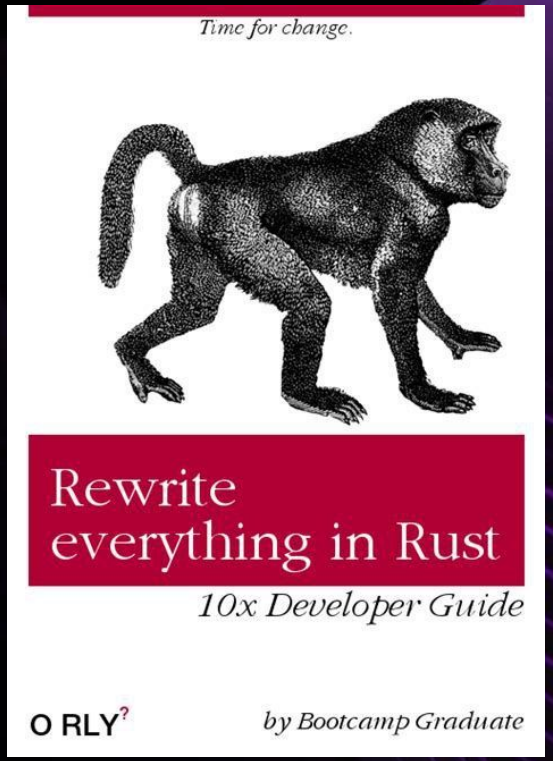
Архитектура



Все это работает в контейнерах для быстрого горизонтального расширения. Соответственно В отдельных контейнерах запускаются инстансы и бд.

Мы задумывали, что будет несколько одновременно работающих сущностей самого хранилища (REST API + Core), которые будут работать с общим бекенд - хранилищем, в котором хранятся пользователи, политики, токены и настройки.

Технологии



Выбор писать часть на питоне, а часть на расте было нашей хотелкой. Потому решили, что будем писать на расте асинхронную прекомпилированную библиотеку для питона (.pyd на вин, .so на линукс).

В качестве базы данных для секретов выбрали MongoDB, потому что она отлично хранит данные формата json, что упрощает работу хранилищам секретов. Для бекенда решили также взять MongoDB, чтобы не внедрять новые технологии, но вариантом лучше будет redis, т.к. схемы для таблиц пользователей, токенов и политик нам известны заранее, а также потому, что in-memory движок в mongodb - платный) Каждый зеленый прямоугольник - отдельно запущенная бд на своем порту

Security features

- Для шифрования использовали AES 256 GCM
- Для создания мастер - ключа Shamir shared key
- Для хэширования паролей использовали Sha256
- Перед помещением данных в бд объект конвертировался в свой зашифрованный вариант, так что все, что покидает программу - зашифровано
- Хотели добавить затирание нулями ключей в памяти и лочить доступ к куче через апи ОС.
- Токены были длиной 128 u8 символов, первые 64 из которых хранились в открытом виде в бд для скорости поиска. При совпадении в хранилище расшифровывается и сверяется вторая часть токена
- Весь трафик (пользователь - ядро, ядро - движок секретов) идет через TLS.
- Алгоритмы и длины токенов и паролей выбраны по стандартам NIST 800 63B

Аудит

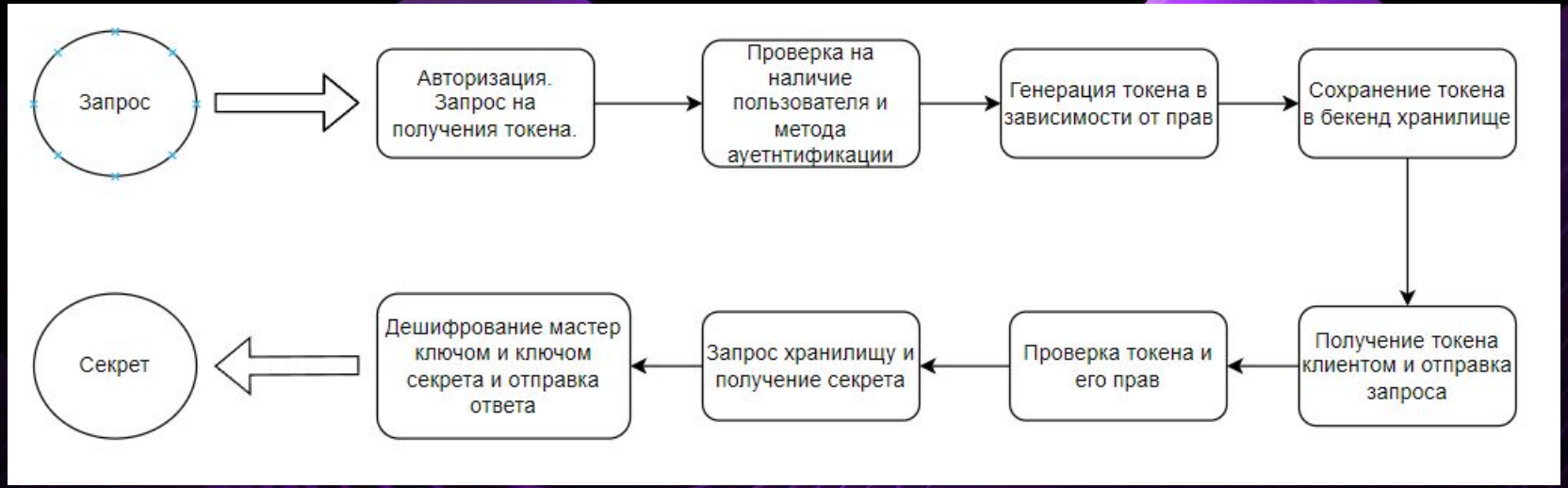
Мы хотели улучшить аудит, логи в системе и реагирование на инциденты путем:

- Ограничение работы root пользователя. Он не может создавать секреты, а также в него нельзя залогиниться через апи, только через консоль.
- Пользователи. Имеют уникальный логин, список доступных методов аутентификации, а также доп данные для быстрой аутентификации пользователя и контакта с ним (в hashicorp vault такого почему-то не было и пользователь был каждый под каждый метод аутентификации)
- У каждого объекта, будь он секретом, пользователем, правилом, настройкой или токеном, есть свой "ответственный" (master). Он вычисляется системой, проверяющей права на совершение действия. Таким образом получится, например, заблокировать удаление секретов пользователями, создателем которых был пользователь X.
- Также хотели добавить возможность настраивать ротацию методов аутентификации и политик к ним. Например, заставлять пользователей менять пароль каждые 2 недели и иметь пароль в 8 символов с обоими регистрами, символами и цифрами.

Серверная часть

Все клиенты с серверной частью взаимодействуют только через REST API. Связь клиента с серверной частью защищена от перехвата TLS шифрованием, так же, как и связь между компонентами решения. Доступ к данным или элементам управления возможен только через аутентификацию/авторизацию. Все запросы обрабатываются в соответствии с применимыми политиками безопасности.

Путь пользователя



Путь пользователя в большинстве совпадает с путем пользователя hashicorp vault. Привилегии прикреплены к токену.

Интеграция

Мы старались повторить API hashicorp vault, поэтому они совместимы насколько возможно.

Текущая архитектура кода не позволяет добавлять свои движки секретов не изменяя исходный код, однако планировалось реализовать подобное без нужды в изменениях.

HA DR

Траффик по REST API распределяется между инстансами ядра с помощью реверс прокси, в нашем случае Nginx.

Данные и кэш хранятся в бд, открывая возможности конкурентной работе нескольких инстансов. Также и несколько инстансов движков секретов могут работать с одним общим хранилищем.

Конечно же планировалась обработка SIGINT, SIGTERM, SIGQUIT для штатного завершения работы.

Также хотели добавить снепшоты, т.к. все хранится в бд, то достаточно сохранить бд и из нее же восстановиться. Поэтому для данной задачи подойдут встроенные в бд снепшоты.

Дальнейшая разработка

Мы не успели внедрить почти все что планировали, т.к. возникли сложности с общей асинхронностью и передачей контекста между питоном и растом.
Мы планируем закончить продукт уже вне хакатона на Rust и добавить безопасную многопоточность, что является его сильной стороной. (может даже работая в ВТБ👉👈)

Код на стадии POC, поэтому в нем много недостатков (например лишних копирований или злоупотребления ? оператора), это также исправим.

Также многие использованные библиотеки дружат с компиляцией в webassembly, поэтому можно превратить наше решение в плагин для браузера или просто удобное веб-приложение.

Еще думали над более пользовательской частью, будет несложно внедрить систему одноразовых секретов в телеграм бота с официальной системой авторизации для сторонних сервисов в качестве метода аутентификации. Так получится сделать защищенный обмен секретами в обход мессенджера, но с его интерфейсом.