MuleSoft

WHITEPAPER

API security best practices

Protect your APIs with Anypoint Platform

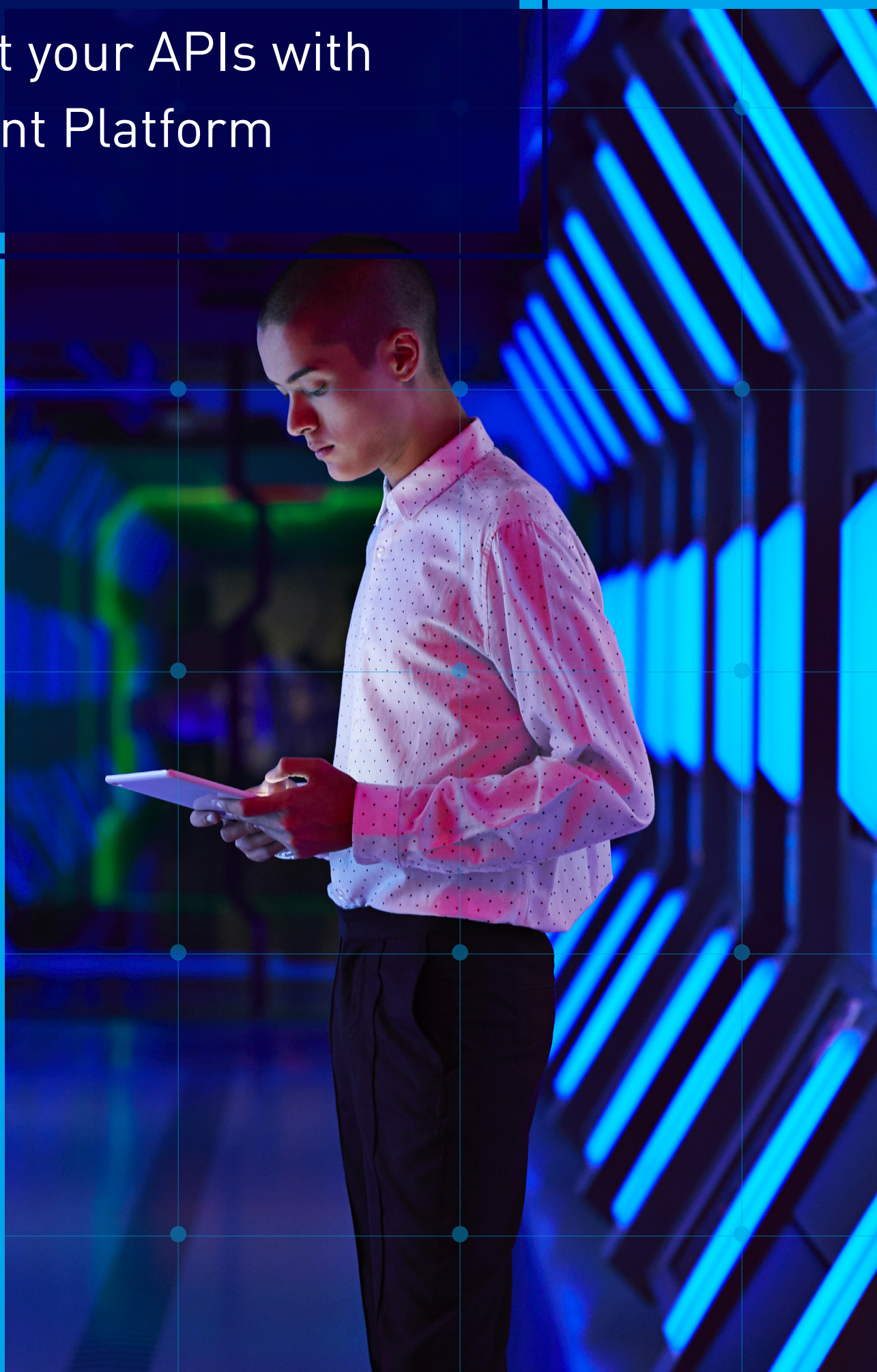

Table of contents

Executive summary

APIs have become a strategic necessity for businesses. They facilitate agility and innovation. However, the financial incentive associated with this agility is often tempered with the fear of undue exposure of the valuable information that these APIs expose. According to Gartner, by 2022, API abuses will be the most-frequent attack vector for enterprise web applications data breaches. It is no wonder that many IT decision makers today are concerned about API security.

MuleSoft's Anypoint Platform can automate the security and governance of your API, ensure your API is highly **available** to respond to clients, and can guarantee the **integrity** and **confidentiality** of the information it processes.

In this paper, we will show how MuleSoft's Anypoint Platform can automate the security and governance of your API, ensure that your API is highly available to respond to clients, and guarantee the integrity and confidentiality of the information it processes. We explore in-depth the main security concerns and look at how the IT industry has responded to those concerns. We present Anypoint Platform as fully capable of managing and hosting APIs that meet strict security standards.

Overview

In this paper, we will discuss APIs and security within Anypoint Platform in two parts. In part one, we will cover the general concerns that senior IT decision makers have with respect to their digital assets.

We will also cover the topics of authentication and authorization, and discuss why it is important to maintain confidentiality, integrity, and availability of your data.

In part two, we will show how Anypoint Platform addresses the above requirements. We will cover the core security capabilities of Anypoint Platform. We will also look at how Anypoint Platform can help you manage your APIs and address your security concerns through policies.

In this part, we will also cover how APIs deployed to MuleSoft's Anypoint Platform can securely integrate with servers in your data center. We will conclude with a fictitious scenario that shows how Anypoint Platform can form part of the fabric of a secure API-led architecture.

Introduction

A secure API is one that can guarantee the confidentiality of the information it processes by making it visible only to the *Users*, *Apps*, and *Servers* that are authorized to consume it. Likewise, it must be able to guarantee the integrity of the information it receives from the Clients and Servers that it collaborates with, so that it will only process information, if it knows that it has not been modified by a third-party.

In this case, the ability to identify the calling systems and their end-users is a prerequisite for guaranteeing these two security qualities. What we have stated also applies to those calls that the API makes to third-party *Servers*. An API must never lose information so it must be available to handle requests and process them in a reliable fashion.

In this paper, we use the term API in a broad sense to include both the interface definition and the service or microservice which implements it. We recognize that many of the standards and examples we present are oriented towards HTTP. However, with our broad definition of the term API, we also envision the use of event-driven approaches with message brokers.

We also utilize the terms: *Users*, *Apps*, *Clients*, and *Servers*. *Users* interact with *Apps* (application software) which are *Clients* to your API. In contrast, your API acts as a *Server* to the *app*. APIs can also act as *Clients* to other APIs, web services, databases, etc.—all of which we refer to as *Servers*. It is a common practice to use the term *messaging* to describe *API calls*.

We utilize both expressions interchangeably for the purposes of this paper.

Section 1: Identity

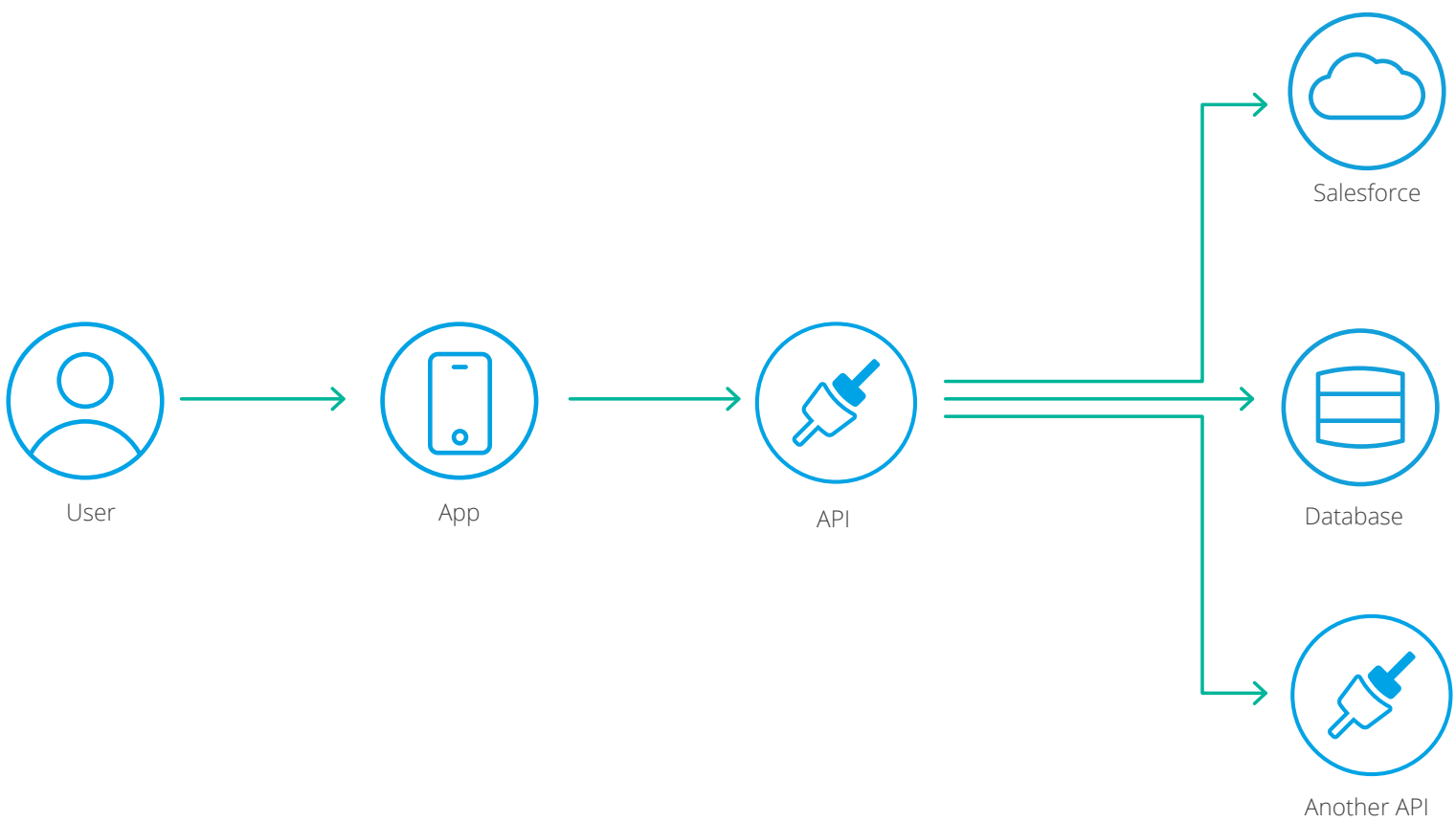

Figure 1: *Users*, *Apps*, *Clients*, and *Servers*

Identity is core to the world of security. You must be able to recognize the *Apps* that consume your API, the Users of the same API, and the *Servers* that your API calls out to. Likewise, your API should be able to identify itself to both *Apps* and *Servers*.

You need an Identity Store that you can refer to verify your recognition of *Apps* and *Users*. The Identity Store could be a database, but an LDAP server is the most popular solution. Active Directory is a popular LDAP implementation. In an LDAP server you typically store usernames, passwords, digital certificates, personal details, and the organization to which *Users* belong. *App* IDs can also be stored here.

An Identity Provider is software that is dedicated to managing the interaction with the Identity Store(s) for authentication and authorization purposes. Your API can function in this role, though it is preferable to delegate this responsibility to the Identity Provider.

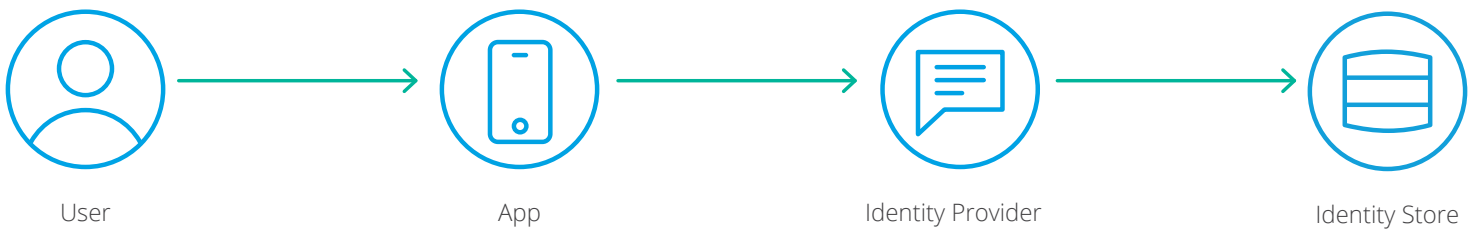

Figure 2: Identity Provider and Identity Store

1.1 User and app authentication

When you are presented with an *App* ID or a *User's* username (claim) in a call to your API, you must be able to verify the authenticity of the claim. This is done with some form of a shared secret. When your API acts as an Identity Provider, it typically authenticates the claim by passing the same credentials to the LDAP server.

1.1.1 Username and password credentials

This is the simplest form of authentication. When it is exposed to *Users*, it places the burden of memorizing the password on them. When it is realized with system-to-system authentication, then a password to a *Server* may end up being shared by multiple other APIs.

- The use of username/password pairs as credentials is a very common practice, but it is not recommended from two perspectives:
  - Passwords have a level of predictability, whereas the ideal scenario is to maximize randomness or entropy. Username/password pairs are a low entropy form of authentication.
  - Maintaining passwords is difficult. If you need to change a password, then you immediately affect all *Apps* that make use of that password. Until each of these has been reconfigured, you have broken communication with them. As a consequence, there is no way you can block

access to one *App* in particular without blocking all the *Apps* that use the same username and password.

1.1.2 Multi-factor authentication (MFA)

Recognizing the weakness of username and password credentials, an *App* using multi-factor authentication (MFA) demands from the *User* a one-time usage token they receive after authenticating with the user's credentials. This token may be delivered through SMS when the *App* requests an MFA Provider to do so. The *User* may also have a digital key which is a token that the *App* can validate. An RSA SecurID is an example of this.

When the *App* receives the token which it validates with the MFA provider, it proceeds to consume your API.

1.1.3 Token-based credentials

A better alternative to Username Password Credentials are token-based credentials, which provide higher entropy and a more secure form of authentication and authorization. The idea is for the Identity Provider to issue tokens based on an initial authentication request with username/password credentials. From then on, the *App* only has to send the token, which greatly reduces the number of username/password credentials going to and from over the network.

Also, tokens are usually issued with an expiration period and can be revoked. Furthermore, because they are issued uniquely to each *App*, when you choose to revoke a particular token or if it expires, all the other *Apps* can continue to use their tokens independently.

In Figure 3, Janet signs into her *App*. The *App* authenticates her and requests a token from the Identity Provider. This authenticates her with the Identity Store and then responds to the *App* with a token. The *App* proceeds to call the API with the token.

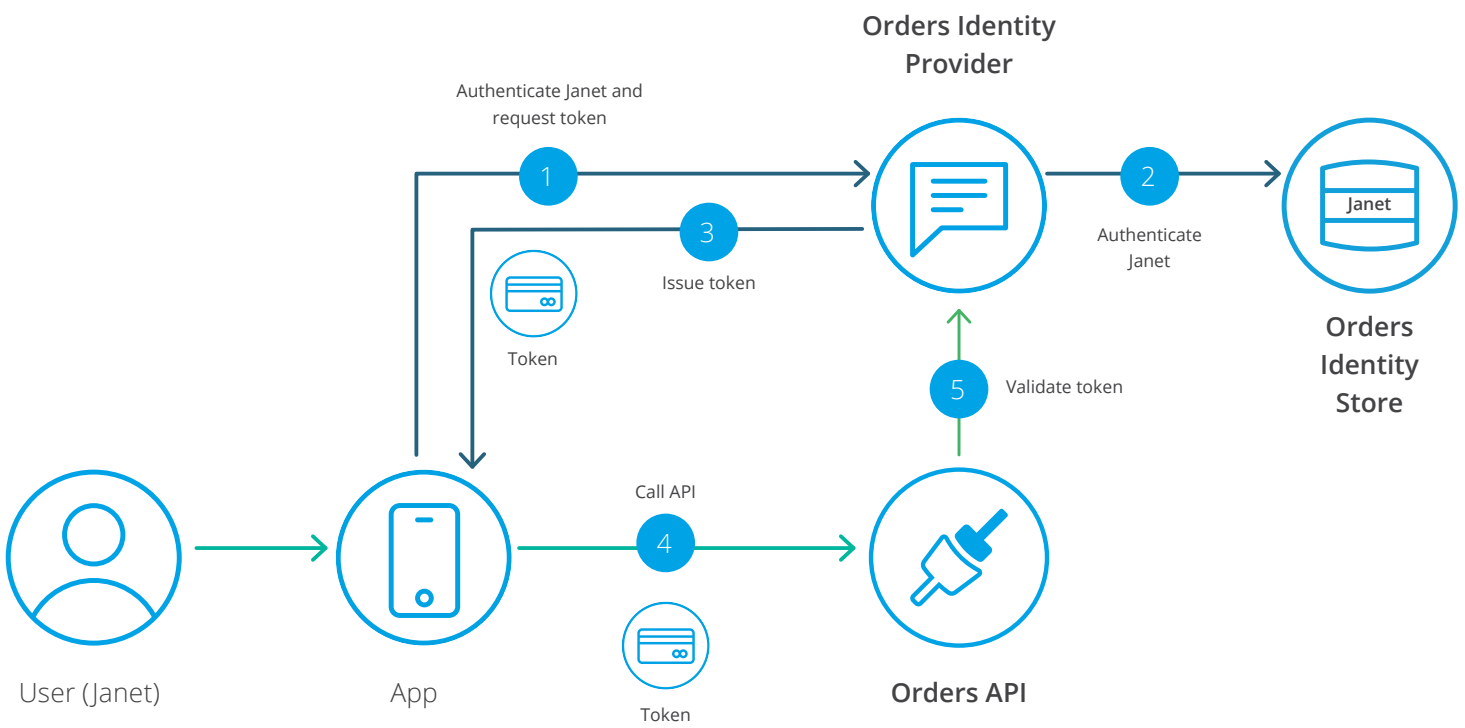

Figure 3: Token-based credentials

1.2 API and server authentication

Your API must be able to authenticate itself to the *Apps* which consume it. Likewise, when your API interacts with *Servers*, they must authenticate themselves to the API. In both cases you must try to avoiding man-in-the-middle attacks, which sometimes take the form of malicious software pretending to be a *Server* or indeed your API.

We will take a look at the typical form of authentication for these scenarios when we address Public key cryptography.

1.3 User and app authorization

1.3.1 Role-based access control (RBAC)

Typically, every business, enterprise, or organization is divided into groups of employees around related business functions. For example, think of the nursing team and the medical doctor team and the catering team in a hospital. The employees working within an organization have a static function defined by these group boundaries.

This group information can be used when software *Users* interact with an *App* and you need to restrict their access

according to the authorization or access control rule in place for that software. You can use the group they belong to identify their role when using the *App*. Groups are role and *App* agnostic. They are purely business-level divisions. LDAP servers use the concept of groups for this purpose. Identity Providers are responsible for retrieving this group information from the Identity Store. A role is an *App* specific definition of access control. A *User* will typically adopt multiple roles defined by each App she uses.

Role-based access control (RBAC) represents a very simple access control mechanism. An *App* need not keep a record of each *User's* level of access to its functions and data. Rather it uses roles to abstract away from those details and assign degrees of access to groups of *Users* which the role represents.

1.3.2 Attribute-based access control (ABAC)

Going beyond the static assignment of roles to *Users* based on the organizational groups to which they belong, attribute-based access control (ABAC) aims to facilitate the dynamic determination of access control based on some sort of circumstantial information available at the time of the API call.

Things like the time of day, the role, the location of the API, the location of the *App,* and combinations of conditions, contribute to the determination of the degree of access. XACML is a standard which defines the rules that must be executed to evaluate the level of access at the time of the API call. The understanding is that this may change from call to call. ABAC often dictates the requirement that your API will respond with subsets of data according to the access control decisions related to the *User*.

1.3.3 Delegated access control with OAuth 2.0

The HTTP-based OAuth 2.0 framework allows an *App* to obtain access to a resource exposed by your API either on its own

behalf or on behalf of the *User* who owns the resource. Thus it allows *Users* to delegate access control to third-party *Apps*.

To facilitate this, your API must collaborate with an OAuth 2.0 authorization *Server*, checking each incoming call for an access token which it must validate with the authorization *Server*. The response from the authorization *Server* will indicate whether the access token is valid (it was issued by the OAuth Provider and it hasn't expired) as well as the scope of access for which the token was issued.

Section 2: Federated identity

The token-based approach to authentication allows for the separation of the issuing of tokens from their validation—thus facilitating the centralization of identity management. The developer of each API needs to only incorporate validation logic within the API so that upon invocation, it looks for the token in the request and then validates it with the centralized Identity Provider. If the token is deemed to be valid (i.e. the *User* or *App* to whom the token was issued has sufficient authorization for this call), then the API should process the call. The parties involved in this call form part of a single security context. In other words, the Identity Provider is able to recognize and authenticate the *App*, the *User* and the API because their identity and shared secret password are in its Identity Store.

However, your API will not always be exposed to an *App* and *User* that the Identity Provider can recognize. What if you wish to expose your API to an *App* in the hands of a *User* from another company, or even from another business unit within your own company? Large companies can have many security contexts with separate identity stores and providers. Federated Identity solves this problem and federated Identity Providers collaborate to facilitate the authentication and authorization of *Users* who belong to different security contexts.

In Figure 4, Janet signs into her *App*. The *App* authenticates her and requests a token from the Orders Identity Provider. This authenticates her with the Orders Identity Store and then responds to the App with a token. The App proceeds to call either the Orders API or the Shipping API. In both cases, the Orders Identity Provider validates the token. The Orders Identity Provider and the Shipping Identity Provider are federated. The Shipping Identity Provider knows that the token has been signed by the Orders Identity Provider to which it delegates the token validation.

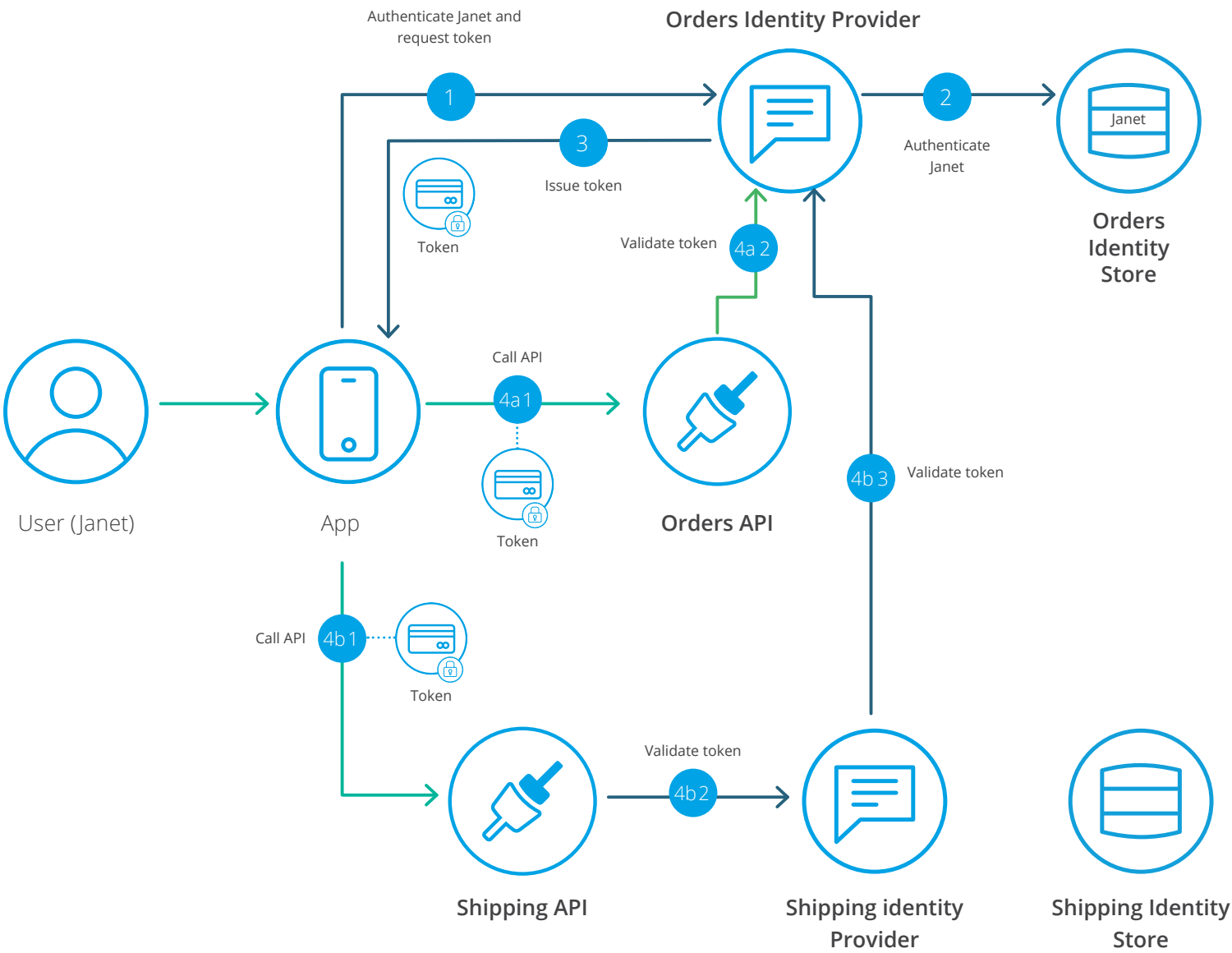

Figure 4: Federated identity

2.3.1 Single Sign-on multi-experience

The Security Assertion Markup Language (SAML) is an industry-standard that has become a defacto standard for Enterprise-level Identity Federation. It allows Identity Providers to communicate authentication and authorization information about *Users* to Service Providers in a standard way.

A SAML Assertion can be issued by an Identity Provider in one security context and be inherently understandable by an Identity Provider in another context. SAML assertions typically convey information about the *User* including the organizational groups to which the *User* belongs, together with the expiry period of the assertion. No password information is provided. The Identity Provider which issues the assertion signs it. The Identity Provider, which has to validate the assertion, must have a trust relationship with the issuing Identity Provider (see Digital signatures).

The primary driver for the use of SAML is within the Enterprise is Single Sign-on (SSO). *Users* don't have to keep separate identities for every application software that they use. Rather, they sign on once with an Identity Provider and from then on any links to applications allow them to bypass the login page of each of these.

Such a setup ultimately delivers the desired *User* experience of not having to maintain multiple sets of username and password credentials and of signing in once, and subsequently bypassing login pages to all of the applications within the enterprise. This SSO experience is usually delivered with a UI Portal which has links to all relevant applications that eliminate the need for any further authentication by the *User*.

SAML can also be used within the context of APIs. We explore this next.

2.3.2 Single Sign-on single experience

The expectation that an Identity Provider in one security context will understand a token issued by an Identity Provider in another security context may very well be reasonable within the boundaries of a particular enterprise. However, this may not be true for external company acquisition scenarios, or for dealing with partners and SaaS facilitating the centralization of identity management.

2.3.2.1 WS-Security with SAML Assertions

WS-Security (in particular WS-Trust) allows Identity Providers to expose SOAP Web Services that will issue identity tokens to requesting *Apps*. A SAML Assertion is one such possible token. The same *App* can then invoke a SOAP Web Service with the SAML Assertion in the header.

WS-Security also caters for the general needs of Integrity and Confidentiality through XML Signature and XML Encryption

(see Message Integrity and Message safety below).

2.3.2.2 OpenID Connect with JWT ID Tokens

OpenID Connect is built on top of OAuth 2.0 to provide a Federated Identity mechanism that allows you to secure your API, in a similar way to what you would get if you exploit WS-Security with SAML. It was designed to support native and mobile apps, while also catering for the enterprise federation cases. It is an attractive and lightweight approach to achieving SSO within the enterprise than the corresponding WS-Security with SAML. Its simple JSON/REST-based protocol has resulted in its accelerated adoption.

Apart from OAuth 2.0 access tokens, OpenID Connect uses JWT (jot) ID tokens, which contain information about the authenticated *User* in a standardized format. Your API can make an access control decision by calling out to a UserInfo endpoint on the Identity Provider to verify if the *User* pertains to a certain role. Just like SAML Assertions, JWT ID tokens are digitally signed (see Digital signatures) so a federated Identity Provider can decide to accept them based on its trust relationship with the Identity Provider that issued them.

Section 3: Confidentiality, integrity, and availability

3.1 Message Integrity

Message Integrity goes beyond the authentication of the App and the User, and includes the verification that the message was not compromised mid-flight by a malicious third-party. In other words, the message received by your API is verified as being exactly the one sent by the *App*. The same goes for when your API acts as *Client* to a *Server*.

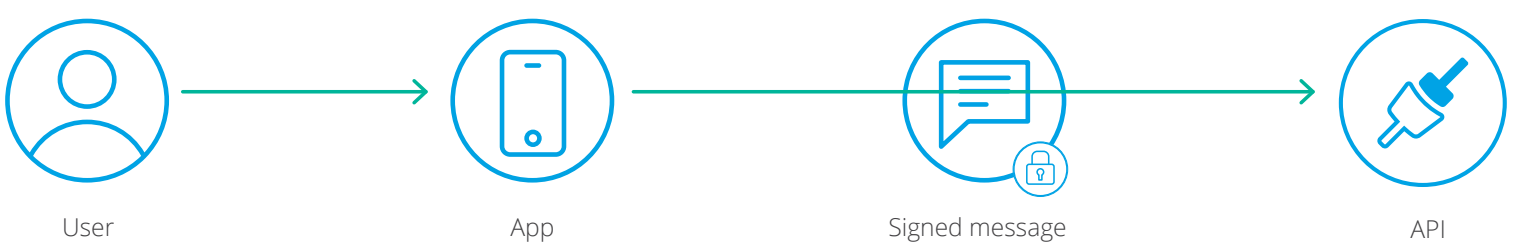

Figure 5: Message Integrity

3.1.1 Digital signatures

We humans sign all kinds of documents when it matters in the civil, legal and even personal transactions in which we partake. It is a mechanism we use to record the authenticity of the transaction. The digital world mimics this with its use of digital signatures. The idea is for the *App* to produce a signature by using some algorithm and a secret code. Your API should apply the same algorithm with a secret code to produce its own signature and compare the incoming signature against this. If the two match, the API has effectively completed authentication by guaranteeing not only that this message was sent by a known *App* (only a known *App* could have produced a recognizable signature), but that it has maintained its integrity because it was not modified by a third-party while in transit. As

an added benefit for when it matters with third-party *Apps*, the mechanism also brings non-repudiation into the equation because neither the *App*, nor the *User*, can claim not to have sent the signed message.

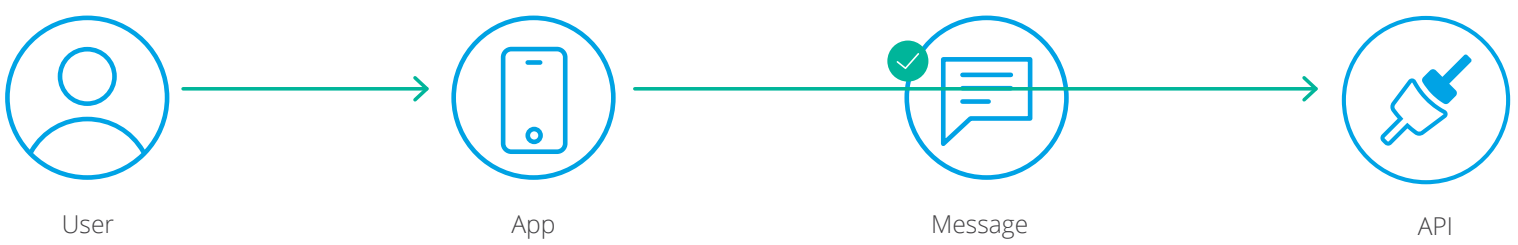

Figure 6: Digital Signatures

3.1.2 Message safety

Even when you know that your API has been invoked by an authenticated *App* and *User*, and the message has arrived with its integrity guaranteed, you still need to protect against any potentially harmful data in the request. These attacks often come in the form of huge XML documents with multiple levels of nested elements. JSON documents may also contain huge nested objects and arrays.

Section 4: Message confidentiality

The integrity of a message sent by a known *App* is assuring, but the journey from *App* to API may have been witnessed by some unwelcome spies who saw all of those potentially very private details inside the message! Thus, it is necessary to hide those details from the point of delivery by the *App* to the reception by the *Server*. An agreement is needed between the *App* and API to be able to hide the details of the message in a way that allows only the API to uncover them and vice versa.

Figure 7: Message Confidentiality

4.1.1 Public key cryptography

The age-old practice of cryptography has made a science of the art of hiding things! IT has adopted this science and can produce an encryption of the message, which is practically impossible to decrypt without a corresponding key to do so. It is as if the *Client* had the ability to lock a message inside some imaginary box with a special key, hiding it from prying eyes until the *Server* unlocks the box with its own special key. *Digital signing* produces signatures in this very way. Cryptography comes in two forms: symmetric, when both *Client* and *Server* share the same key to encrypt and decrypt the message; and asymmetric, when the *Server* issues a public key to the *Client* allowing the *Client* to encrypt the message, but keeps a private key which is the only one that can decrypt the message: one key to lock the message and another key to unlock it.

4.1.2 Digital Certificates

Digital Certificates are a means to facilitate the secure transport-level communication (TLS) between a *Client* and a *Server* over a network in such a way that the *Server* can authenticate itself to the *Client*. This is made possible because the certificate binds information about the *Server* with information about the business, which owns the *Server*. The certificate is digitally signed by a Certificate Authority which the *Client* trusts.

4.1.3 Mutual authentication with Digital Certificates

In most cases, it is the *Server* which authenticates itself with the *Client*. However, there are also scenarios in which the *Server* demands the authentication of the *Client*. The *Server* requests the *Client* certificate during the TLS handshake over the network. One thing to keep in mind is that the *Server* controls whether *Client* authentication occurs; a *Client* cannot ask to be authenticated.

Mutual authentication with TLS certificates is ideally suited to the type of system-to-system communication that you see when your API acts as a Client to a Server, whether the Server be another API or a database or any other system entity.

Missing from this communication is the human *User*. Hence, the security credentials exchanged between the two parties are far easier to manage.

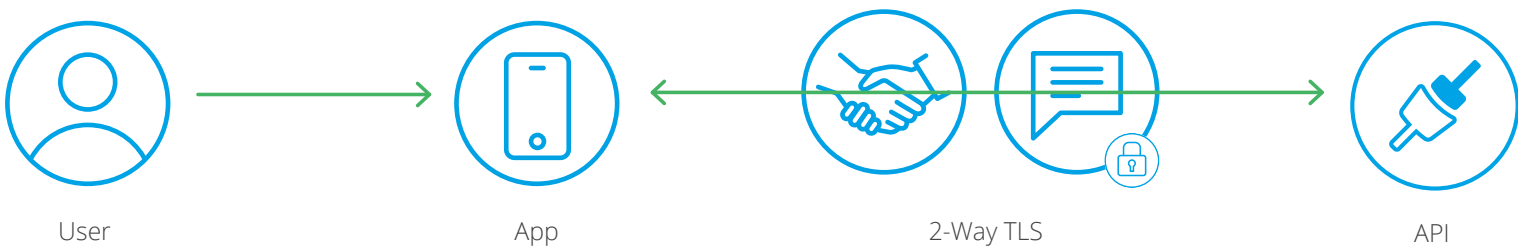

Figure 8: Mutual authentication

4.1.4 HTTPS

By utilizing TLS, your API can expose itself over HTTPS and guarantee both message integrity and confidentiality at the same time. Public keys are emitted on certificates which have been digitally signed by independent and trusted certificate authorities, thus guaranteeing that the public key was issued by your API. Once the initial handshake has been completed with the *App* by the exchange of messages using public and private keys, the communication switches to the more efficient symmetric form using a shared key generated. This is just for the duration of the communication, all of which occurs transparently.

4.1 API availability

Your API must guarantee that it is always available to respond to calls and that once it begins execution on the call, that it can finish handling the received message immediately without losing data. This can be achieved by horizontally scaling the API across multiple servers and by handing off the processing of the message to a message broker, which will hold the message till the API has completed its processing. The understanding in this latter scenario is that another process is subscribed to this message publication and thus continues the processing asynchronously.

It is clear that *reliability* is a step beyond mere *availability*. While an API (through horizontal scale-out) may be available to respond to all calls (because a load-balancer in front of the API can propagate calls to the correct hosting server when any of the other servers are down), this still may not be enough because the API may fail mid-way through processing. In a reliable architecture, the API would receive the call and then leave a message on a message-broker queue (JMS or AMQP for example). Even when the service subscribed to the queue is down, the broker can hold onto the message for later consumption when the service comes back online again.

Section 5: Mule runtime security capabilities

Mule runtime addresses a broad set of security concerns with best practice solutions for transport and message-level security. Your API can be hosted by Mule, exposed over HTTPS and can facilitate TLS client authentication. Likewise, your API can make calls out to servers over HTTPS and issue client certificates as needed. Keystores and truststores are used to store TLS certificates for these scenarios. Messages that are sent to exchanges and queues on Anypoint MQ, MuleSoft's cloud messaging solution, can be encrypted. Mule runtime can also encrypt and decrypt messages, digitally sign them, and verify the validity of incoming digital signatures. IP white- and black-listing is also available.

With these capabilities, Mule runtime addresses the concerns of exposing highly available APIs that authenticate and authorize incoming calls, while guaranteeing message integrity and confidentiality.

5.1 Message confidentiality on Mule runtime

5.1.1 MuleSoft's HTTPS Connector

An HTTPS listener can be configured with reference to a keystore so that your API can authenticate itself to the *App*. When the *App* demands client authentication, the listener can reference a truststore. Similarly, when your API needs to interact with a *Server* over HTTPS, you can use an HTTPS requester, which references a keystore to authenticate itself and a truststore for when digital certificates are not recognizable by the standard Java JDK truststore (cacerts).

5.1.2 Mule Encryption Processor

The Mule Message Encryption Processor can change the content of a message so that it becomes unreadable by

unauthorized entities. Mule can encrypt the entire payload of a message or specific parts of the payload, according to security requirements, using different encryption strategies.

5.1.3 Dynamic data filtering with DataWeave

For ABAC scenarios (see Attribute-based access control (ABAC) you will need to filter the payloads that your API sends back to *Apps* based on the degree of access determined either for the *App* or for the *User*. DataWeave is Mule's data transformation engine, which transforms between different mime-types using a simple expression language which is common across all data formats.

The language can be used to remove and/or mask data fields in the payload, whatever the structure of the payload and the location of the field. The same expression can be stored in a datastore and 'blindly' executed by the engine at runtime. In this way, you can cater for dynamic transformation logic that you may not necessarily be able to configure at design time. Rather, based on criteria decided at runtime, you can choose a transformation and apply it to the payload.

5.2 Message integrity on the Mule runtime

Exposing your API over HTTPS guarantees that it has not been modified in transit. However, authentication and authorization of the request still need to take place.

5.2.1 Mule Security Manager

Central to authentication in Mule is Mule Security Manager. This is the bridge between a standard Mule configuration and Spring Security beans.

Figure 9 illustrates how credentials are passed and validated in the solution. The security-manager, as you can see below, delegates to the authentication-manager. The authentication-manager uses the authentication-provider to authenticate the

set of credentials. The authentication-provider abstracts away from the details of the system used to do the authentication, whether it be in-memory, LDAP, or DB-based. The Spring LDAPAuthenticationProvider uses the BindAuthenticator to build a DN-based on the credential username to bind directly to the LDAP server.

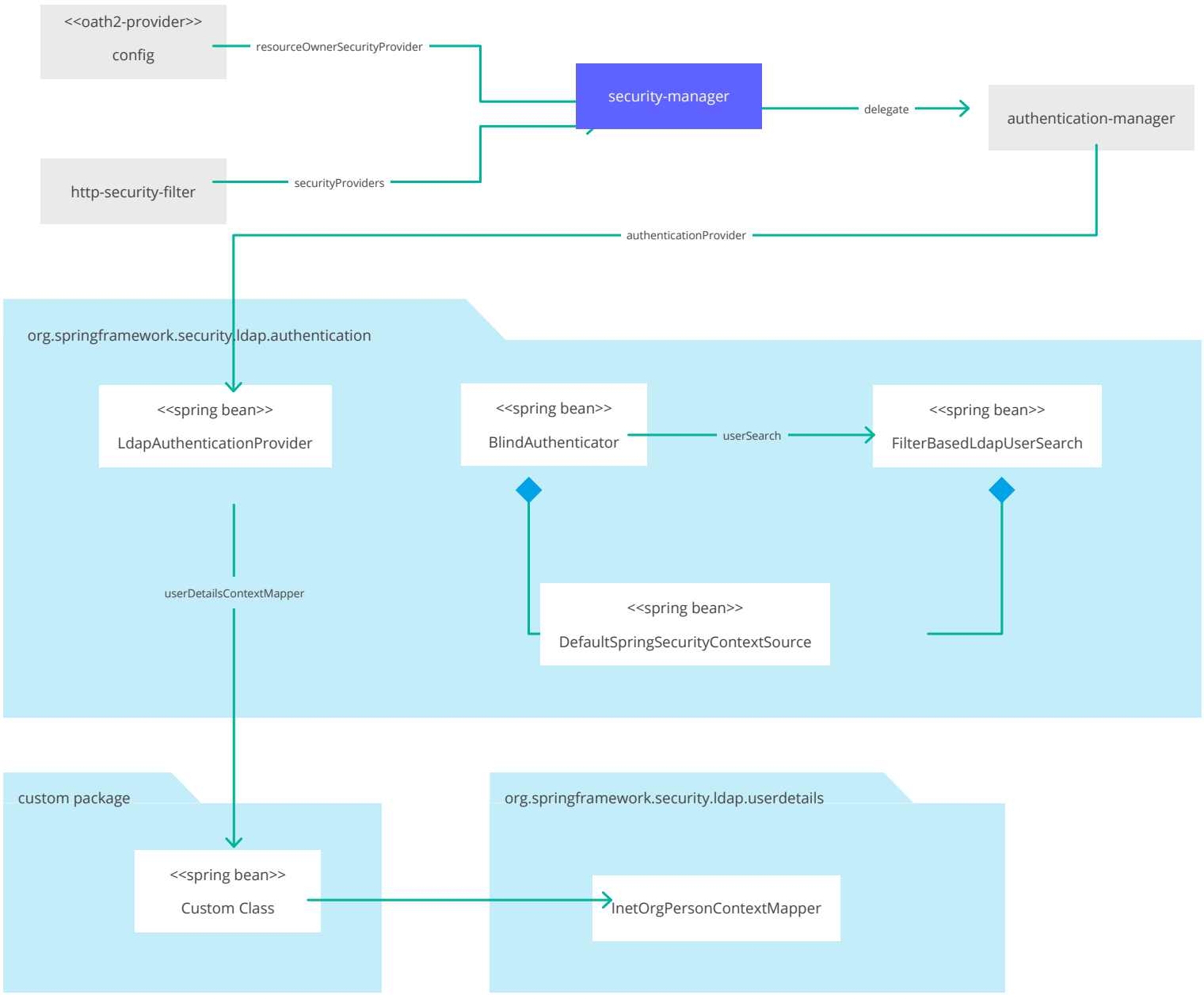

Figure 9: Passing and validating credentials

5.2.2 Mule Secure Token Service OAuth 2.0 Provider

Mule can act as an OAuth2 Provider, issuing tokens to registered *Apps*, applying expiration periods to these tokens and associating them to *User* roles and fine-grained access control known in the OAuth world as scopes. Refresh tokens can also be issued and tokens can be invalidated. Mule can subsequently validate incoming tokens against expiration periods, roles, and scopes and thus grant or deny access to the

flows in the Application. Scopes represent broad levels of access to the Mule flows. The provided access token must be sent in with each request and can be validated by Mule to ensure it has not expired or been revoked and that it has the scopes that correspond to a particular flow.

More fine-grained control can also be applied by comparing the role of the *User* for whom the token was issued with the allowed roles for the flow. The validate filter has a resourceOwnerRoles attribute to specify these. The granularity of access control can be in either the grant or the role.

As we extend APIs outside of our organization, we may have to cater for applications belonging to partners. Imagine we were to dynamically expose access to your API to a mobile application. We need only register this new client in your OAuth 2.0 Provider configuration.

5.2.3 Mule Digital Signature Processor

Mule Digital Signature Processor adds a digital signature to a message payload, or part of the payload, to prove the identity of the Message's sender. Mule can also verify a signature on a message it receives to confirm the authenticity of the Message's sender.

5.2.4 Mule Credentials Vault

Mule Credentials Vault is for the encryption of properties, which are referred to and decrypted by the Mule application at deployment time. These properties are encrypted with a variety of algorithms and are completely hidden from anyone who does not have the key to the credentials vault. At deployment time, the key is passed to Mule as a system property. This key should only be in the hands of authorized personnel.

5.3 API availability with Mule HA clusters

A single Mule server hosting your API is not enough to facilitate high availability. To achieve this, you need to host the same API on more than one Mule runtime. With a load balancer in front of the API, you can guarantee that the API will always handle incoming requests as the load balancer chooses between instances that are healthy.

Reliability on Mule can be achieved by clustering two or more instances of Mule together, which is easy using Anypoint Runtime Manager. In this scenario, we configure the Mule VM endpoint as a reliable handoff mechanism immediately after receiving the message. Another flow processes the message from that same VM endpoint. If the Mule node which receives the Message from the VM goes down, then another Mule node on the same cluster will pick up the same Message.

For reliable processing of Messages between multiple APIs you can use Anypoint MQ (see Anypoint MQ).

Section 6: Anypoint Platform security capabilities

6.1 Anypoint API Manager and Anypoint Security

Anypoint Platform is a fully multi-tenant solution running on top of Amazon Web Services (AWS) and inside a VPC (cloud VPN). Data, metrics, and metadata cannot be accessed across organizations.

Although Anypoint Platform can manage and enforce the runtime security of your API, the API itself remains wherever you have deployed it. Only the configuration of the policies, metadata about your API, and analytics about the usage of your API is stored in Anypoint Platform.

6.1.1 API adaptability through policies on API Manager

API management is a discipline which addresses the need to publish your API for consumption by known *Apps*, registering those *Apps* and provisioning them with their own ID, and secret identifiers, authorizing the *Apps* to consume your API, and adapting the APIs to the potentially different security requirements across the *Apps*.

The adaptability is addressed with what we call policies. These are encapsulations of the types of logic that usually recur across your APIs. Similar to how aspect-oriented programming worked, these logical bundles can be applied to or removed from running APIs without affecting their lifecycle. Security is a prime example of such logic. We explore security policies next.

6.1.2 Secure communication between the Mule runtime and API Manager

Using the Mule runtime as an API gateway, you can host your API. Mule runtime communicates constantly with Anypoint API Manager and Anypoint Security to retrieve policies and report back analytical information about the usage of your API. This communication is initiated by Mule runtime, which authenticates itself with OAuth 2.0 *Client* Credentials. You configure Mule runtime with a client ID and secret which is configured for your particular organization (or business group) in Anypoint Platform. The ID and secret are used by Mule runtime to get an OAuth 2.0 token to be used in subsequent calls. All calls are to a RESTful service which is accessible over HTTPS. Mule runtime is insulated from external network outages since it stores a local cache and can continue to operate even if Anypoint Platform were to become unavailable. Regardless, MuleSoft maintains an SLA of 99.99% for Anypoint Platform. Anypoint Platform is certified via WhiteHat Sentinel.

6.1.3 API security policies

We must return to our discussion about identity in the light of what Anypoint Platform has to offer in its suite of policies. Some of the following policies are inherently dependent on a mechanism to verify incoming Identity tokens. All of them address security concerns:

- **Client ID enforcement:** Locks down your API for consumption by only a set of known clients.
- **SLA-based Rate Limiting:** Provides different quality of service contracts to your known clients, 10 calls a minute for some, 100 calls a second for others, etc.
- **SLA-based Throttling:** Same concept as Rate Limiting, only exceeded calls are queued for next time window.
- **Mule OAuth 2.0 access token enforcement:** Validates incoming tokens previously issued by Anypoint OAuth Provider upon receipt of client ID and secret.
- **External access token enforcement:** Validates incoming tokens previously issued by PingFederate or Open AM OAuth Provider upon receipt of client ID and secret.
- **LDAP Authentication policy:** Authenticates using the configuration details for an Open LDAP or Active Directory LDAP which are already configured in your enterprise.

- **Cross-origin resource sharing (CORS):** Permits your API to be invoked by a JavaScript client (an Angular App, for example) which is hosted on a domain different from your API.
- **HTTP Basic authentication:** Authenticates using credentials which are configured in a security manager underlying this policy.
- **IP blacklist and whitelist:** Denies or permits calls only from IP addresses present in this list.
- **JSON and XML threat Protection:** Guarantees the safety of the messages passed to your API.
- **Spike Control:** Ensures that within any given period of time, no more than the maximum configured requests are processed.
- **Header Injection and Header removal:** Adds or removes specified headers to the request/response of the message.
- **Tokenization and De-Tokenization:** Protects sensitive data with vaultless, format-preserving encryption or masking.
- **JSON Web Token (JWT) validation policy:** Validates incoming requests using a JWT with JWS format, verifies the signature of the token and asserts the values of the claims.

Mule runtime stores the client IDs and secrets of consuming *Apps* in an Identity Store. When you register a new consuming App in the API Portal, Anypoint Platform generates a new *App* ID and secret and persists it. Later, when any identity-related policy, like SLA-based throttling, is applied to your API, Mule runtime downloads the policy and also downloads the ID and secret for every consuming *App* registered to consume your API.

Thus, when Mule runtime injects the Policy configuration into your API, it also provides access to a local embedded database of IDs and secrets which the Policy consults to verify the identity of the calling *App*. When you choose to integrate your Anypoint organization with an external identity management

technology like PingFederate, this assumes the role of administering and persisting *App* ID and secrets.

6.1.4 Custom security policies

Policies on Anypoint Platform are snippets of Mule Configuration. As such, custom policies are easily configurable and can be surfaced on the API Manager portal as siblings to our out-of-the-box policies.

We have a number of custom policies published to Anypoint Exchange. These cover SAML-based security use cases such as the ability to validate incoming assertions, or username tokens.

You can write your own policy to cover any area of logic that is pertinent to your API. For the ABAC scenarios that we described above, you might consider configuring a custom policy which accepts either a full DataWeave expression in the API Manager portal UI at policy application time. This can be executed at runtime on the response payload. (See Dynamic data filtering with DataWeave).

If you wish to protect your API with OpenID Connect, you should consider writing a custom policy to validate incoming tokens against the Authorization *Server*.

6.1.5 API products

Certain API security policies, such as SLA-based rate limiting and access control enforcement, can be applied to individual APIs, across a collection of APIs or both. This provides you with flexibility in how you promote, provision, and manage access for APIs that may be commonly consumed together.

6.1.6 Automated policies

Any API security policy can also be configured to be automatically enforced across every API deployed within a given environment. Common scenarios for automated policies

include authorization and rate limiting. Automated policies can also be audited for compliance via a simple reporting API call.

6.1.7 Edge security policies

Anypoint Platform provides an additional layer of protection for APIs through Anypoint Security. The Edge gateway can be configured to enforce security controls on all APIs deployed within a given perimeter. The policies described below can then act as a default router capability through which all traffic traverses.

- **Denial of service (DoS):** Protects APIs against malicious clients trying to flood your network to prevent legitimate traffic to your APIs.
- **IP whitelist:** Create an IP address whitelist policy to configure an explicit list of IP addresses that can access your deployed endpoints.
- **HTTP limits:** Prevent attacks from clients that send large messages that can consume all of your processing bandwidth.
- **Web Application Firewall:** WAF policies provide the Open Web Application Security Project (OWASP) Core Rule Set (CRS) for checking requests and responses to detect common web application attacks.

You can use Anypoint Security policies to manage all traffic to your APIs, and leverage API Manager policies to apply specific behaviors to specific APIs.

6.1.8 Feedback loop between edge gateway and API gateway

After you implement security in this way with Anypoint Platform, a feedback loop between the Edge and API gateways will automatically detect API attacks, escalates them to the perimeter, and updates protections to eliminate vulnerabilities. This allows you to automatically enhance security with a learning system that adapts as new threats emerge.

Section 7: Anypoint MQ

This is a multi-tenant cloud messaging service offering persistent data storage across multiple data centers, ensuring that it can handle data center outages and have full disaster recovery. For compliance with your data at rest policies, Anypoint MQ allows you to encrypt all messages that arrive in either an exchange or a queue.

7.1 Anypoint Platform Virtual Private Cloud

Mule applications can be deployed either to your on-premises Mule runtime or to our fully-hosted and fully-managed iPaaS. In most scenarios, Mule applications deployed to the iPaaS will need to integrate with systems in your datacenter. In some cases a hybrid architecture is adopted where Mule applications deployed to the iPaaS must integrate with Mule applications deployed to Mule runtime on-premises. Either way, there is a need to establish a secure network between the cloud and your datacenter.

Virtual Private Cloud (VPC) enables you to connect your organization in Anypoint Platform to your corporate data centers – whether on-premises or in other clouds – as if they were all part of a single, private secured network. You can configure these networks at hardware or software levels.

VPC can be configured to use IPSec, TLS (over OpenVPN), or Amazon VPC peering to connect to your on-premises data centers. IPsec connections can be configured at the hardware-level in addition to a software client. If you already use Amazon, you should use VPC peering. Otherwise, IPsec is in general the recommended solution for VPC to on-premises connectivity. It provides a standardized, secure way to make connections and integrates well with existing IT infrastructure such as routers and appliances.

7.2 Anypoint Platform user roles and permissions

In Anypoint Platform, *Users* belong to an organization and have a set of roles and permissions. API versions and deployment environments are grouped under organizations (and optionally under Business Groups too), to access them you need to have an account that owns the necessary permissions and that belongs to its corresponding organization – and to the Business Group if the resource exists in one.

Roles and permissions can be granted for accessing resources that exist in the master organization, or for resources that exist within a Business Group. A *User* that owns any role of a business group is implicitly granted membership in the Business Group.

Each role contains a list of permissions that define what a *User* that holds that role can do with the specific resources it scopes. Permissions can also be added at an individual user-level, without the need for roles. There are two different types of permissions: those that are for API versions and those that are for iPaaS environments. Keep in mind that API permissions are API version specific and iPaaS permissions are environment specific – they grant you the ability to do something within a particular API version/environment, not the entire organization. The only exceptions to that rule are the roles API Versions Owner – which grants ownership of all APIs and all of their versions within the corresponding Business Group – and Portals Viewer – which grants viewing access to all portals in the corresponding Business Group.

7.2.1 Federated user access to Anypoint Platform

Anypoint Platform can be integrated with your organization's external federated identity provider. Opting to use federated identity management for Anypoint Platform gives your *users* single sign-on access and facilitates OAuth security for APIs using the same identity management system.

Anypoint Platform supports SAML 2.0 identity providers for *User* management, the following ones were successfully tested working with the platform:

- PingFederate
- OpenAM
- Shibboleth
- Okta
- ADFS

You can set up your Anypoint Platform organization so that when a SAML *User* belongs to certain groups, it will automatically grant certain equivalent roles in the your Anypoint Platform organization.

Section 8: Anypoint Platform compliance

When Anypoint API Manager manages APIs from the cloud, it stores only metadata about the APIs and the *Apps* which consume them. The APIs can be deployed on the Mule runtime either on-premises or in our fully hosted, fully managed iPaaS solution. This fully managed solution meets rigorous industry standards that have been validated through the work of MuleSoft's compliance team and external auditors.

MuleSoft uses the ISO 27001 framework for our security program. We also have been assessed for the following industry standards: ISO 27001, SOC 1 Type 1, SOC 2 Type 2, PCI-DSS, and FedRamp. All compliance reports are shared with our customers under a non-disclosure agreement (NDA).

8.1 Data privacy

MuleSoft understands that data is the most important asset of every company. We are focused on ensuring the privacy and security of customer data at all levels. Our privacy policy describes how we collect, process, and disclose personal data, and our customer data protection policy lays out the robust security and data protection mechanisms we have put in place to ensure the security of data across our products and services. We abide by laws and regulations wherever we operate, including the European Union's General Data Protection Regulation (GDPR). As a company, we have certified compliance with the EU-U.S. and Swiss-U.S. Privacy Shield Framework. To further aid customers in compliance with data residency requirements, MuleSoft provides a fully hosted version of Anypoint Platform within the EU through data centers in Germany and Ireland. Furthermore, MuleSoft customers can select the country in which their runtimes will reside.

Section 9: Summary scenario

Let us consider a hypothetical scenario: Mythical Retail have a chain of stores and deliver an eCommerce solution to their customers. One of their business objectives is to increase their revenue by 20% over the next 18 months. To achieve this goal, they aim to improve customer loyalty by providing a compelling omnichannel digital experience. This experience will allow customers to shop with ease wherever and whenever they want, receive appropriate recommendations and offers, and see their current loyalty balance in real-time. Mythical Retail want a 360-degree view of their customers' visits across all touchpoints. They aim to reward every customer interaction made online and in-store and have a clear view of their customers' spending habits.

Mythical Retail has adopted MuleSoft's API-led connectivity approach to integration and see APIs as strategic assets upon which they can execute digital initiatives. They have invested in clienteling software which can help their sales associates meet and register anonymous customers in-store and identify customers already registered. This software will also provide a mobile point-of-sale (POS) experience. With APIs backboning the clienteling solution as well as the customer's web and mobile interactions, Mythical Retail can guarantee the uniformity of the customer experience across every touchpoint.

9.1 Securing customer transactions

Katie is a customer of Mythical Retail and likes to shop online. She uses the iPhone *App* to make orders. Katie registered with Mythical Retail through a sales associate in-store. Her details are stored in Active Directory.

Figure 10 is an overview of the APIs and *Servers* needed to deliver her the capability to place an order on the phone.

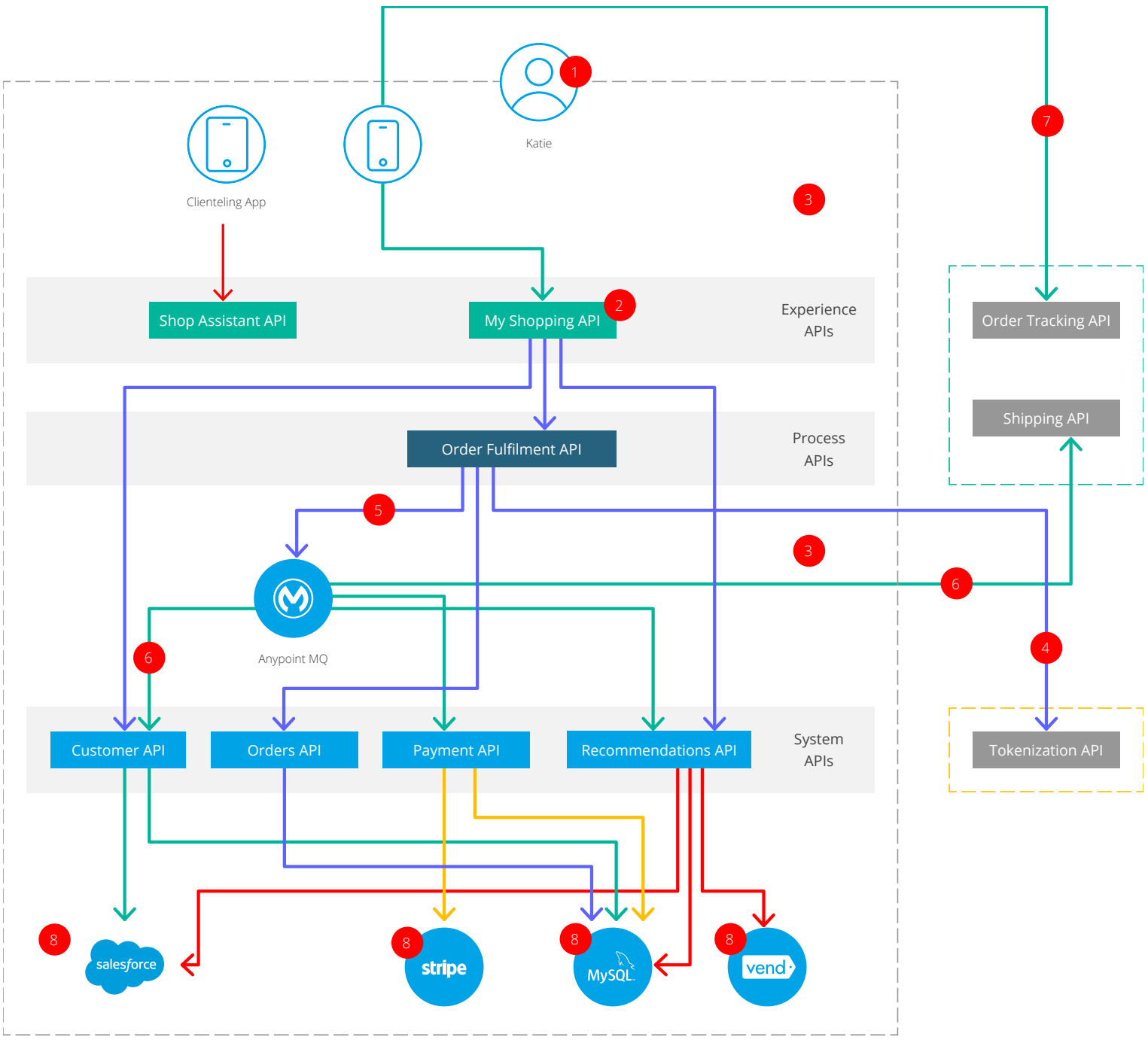

Figure 10: An overview of the APIs and servers needed to place an order on a phone

1. Katie must sign in to the *App*.
2. The *App* must authenticate itself on Katie's behalf and consume the My Shopping API with the relevant degree of access control. Only Katie's data must be accessible for this interaction.
3. All calls between experience, process, and system APIs must be protected.
4. The Order Fulfillment API orchestrates the Orders API and the Tokenization API. The latter is exposed by a third-party credit card processing company and delivers obfuscation functionality to Mythical Retail so that Katie's credit card details are never stored in their original form in the systems of record. The call to the Tokenization API needs to be authenticated, signed, and encrypted.

5. The Order Fulfillment API creates a business event which it publishes to an encrypted exchange on Anypoint MQ. The published data must be encrypted and the publication call must be signed, encrypted, and authorized.

6. The Customer API subscribes to the event on Anypoint MQ and is responsible for increasing Katie's loyalty points. The Recommendations API is also registered to consume this event and gathers the details of the order to feed into future recommendations accordingly. Likewise, the Payment API subscribes to the event and finalizes the financial transaction. The shipping of orders is the responsibility of Mythical Retail's partner. Their Shipping API also subscribes to the same event. All of these subscriptions to Anypoint MQ must be signed, encrypted, and authorized.

7. The iPhone *App* utilizes the Order Tracking API, which is exposed by Mythical Retail's partner. The partner forms a separate security context and Katie is not in their Identity Store. Her claim must be recognizable in the shipping context.

8. Interaction with the systems of record must be secured according to the requirements of each *Server*.

9.2 Anypoint Platform as part of the security fabric for Mythical Retail

Figure 11 is an overview of the APIs used to deliver the loyalty experience to Mythical Retail's customers. All the APIs are deployed to MuleSoft's iPaaS and managed by Anypoint API Manager. Anypoint MQ is used for messaging between the APIs and PingFederate is used as Identity Provider, MFA Provider, and OpenID Connect/OAuth Provider. All HTTP APIs in this context are protected with HTTPS and API security policies are applied to each of the APIs using Anypoint API Manager.

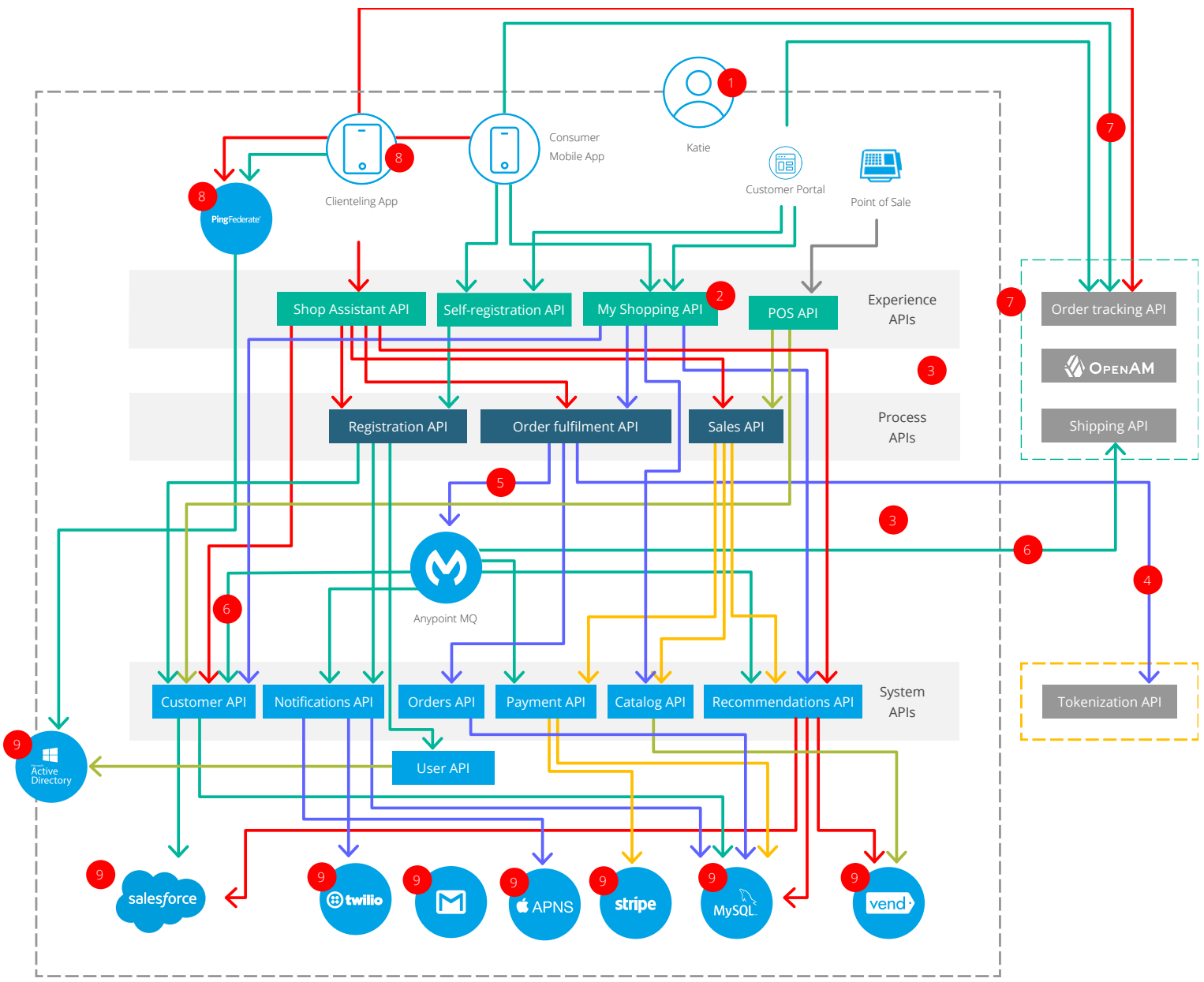

Figure 11: An overview of the APIs used to deliver a loyalty experience

1. Katie signs in to her iPhone *App*.

2. Part of the digital solution catering to Katie's experience is the My Shopping API which delivers all the relevant capabilities to her iPhone *App*. This API is protected with the PingFederate access token enforcement policy. To consume the My Shopping API, the iPhone *App* must interact with the PingFederate Authorization *Server* to request an OpenID Connect token on her behalf. PingFederate authenticates her credentials against Active Directory. Upon successful authentication, it generates a token and signs it (see Digital signatures) before responding to the iPhone *App* with the token. The *App* presents this token in a custom HTTP header on every subsequent call to My Shopping API.

3. All calls between experience, process, and system APIs are protected with the client ID enforcement policy. Each API has an ID and secret stored in Mule Credentials Vault.

4. The calls to the Tokenization API are protected with client cert authentication and the payload passed to the API is encrypted and signed with Mule Encryption Processor and Mule Digital Signature Processor respectively before sending.

5. Publications to Anypoint MQ are protected with OAuth 2.0 and HTTPS.

6. Subscriptions to Anypoint MQ are protected with OAuth 2.0 and HTTPS.

7. There is a federated trust between the Identity Providers in both security contexts for Mythical Retail and its shipping partner. Both the sales associate's *App* and Katie's *App* can call the Order Tracking API with the access token that they received from PingFederate. The Order Tracking API validates the token with OpenAM, the Identity Provider of the shipping company. This can verify that the token was signed by the Identity Provider of Mythical Retail, which it trusts. The Order Tracking API accepts the invocation by the iPhone *App* and responds accordingly.

8. Sales associates use multi-factor authentication to sign in to their *App*. The *App* calls PingFederate OAuth 2.0 authorization server to get a token which it passes to the Shop Assistant API.

9. Interaction with the systems of record is secured in various forms according to the requirements of each Server. Tokens and username and password credentials are stored in Mule Credentials Vault.

Conclusion

APIs are a strategic necessity to give your business the agility and speed needed to succeed in today's business environment. But with the increasing cost of security breaches, senior IT decision makers quite rightly want assurances that exposing their data via APIs will not create undue risk. Anypoint Platform can automate the security and governance of your API, ensure that your API is highly available to respond to clients, and can guarantee the integrity and confidentiality of the information it processes.

About MuleSoft

MuleSoft, a Salesforce company

MuleSoft's mission is to help organizations change and innovate faster by making it easy to connect the world's applications, data, and devices. With its API-led approach to connectivity, MuleSoft's market-leading Anypoint Platform™ empowers over 1,600 organizations in approximately 60 countries to build application networks. By unlocking data across the enterprise with application networks, organizations can easily deliver new revenue channels, increase operational efficiency, and create differentiated customer experiences.

For more information, visit **mulesoft.com**