WHITE PAPER | 2014 年 11 月

OAuth と API のセキュリティに関するハウツー・ガイド

企業の OAuth 実装を容易にするには

ca technologies

目次

OAuth について

OAuth は、アプリケーションやデータへの限定的なアクセスを認可する新しい Web 標準です。これを使用すると、ユーザは自分が所有するリソースへの限定的なアクセスを許可できるように設計されています。写真の印刷サイトなどのサードパーティ・クライアントに対して、Flickr や SmugMug などのサイト上に保管している写真へのアクセスを許可する場合などです。以前はユーザ名とパスワードをクライアントと共有するようユーザに依頼することがよくありましたが、これは一見単純な要求のように見えて、受け入れがたいセキュリティ・リスクを覆い隠しています。これと異なり OAuth では最小限の特権モデルを推奨しているため、ユーザは限定的な機能のトークンを発行して自分のアプリケーションやデータへの限定されたアクセスを許可することができます。

OAuth は Web の管理の委譲を実際のリソース・オーナーの手に委ねるため、重要です。異なる Web アプリケーション上のアカウントをユーザが結び付けるため、それぞれのサイトのセキュリティ管理者が直接介入する必要はありません。この関係は長く存続させることもでき、また、いつでも簡単にユーザが終了することができます。OAuth が WEB コミュニティにもたらした最大のメリットの 1 つは、アイデンティティ・マッピングをユーザに委譲するプロセスを形にしたことです。

OAuth は急速に現代の Web の基本標準になりつつあり、もともとの出発点であるソーシャル・メディアをはるかに超えて発展しています。今では OAuth は企業に取って非常に重要になりました。保険会社やケーブルテレビ会社、医療機関さえも、リソースへのアクセス管理に OAuth を使用しています。OAuth の導入の多くは、ますます多様化するクライアントや特にモバイル・デバイスを企業がサポートしなければならないことが要因となっています。こうした企業はこの新しいデリバリ・チャネルにサービスを提供するために API を積極的に導入しており、OAuth は API の認可のベストプラクティスです。

ただし、OAuth は完全な API アクセス制御とセキュリティ・ソリューションの構成要素の 1 つにすぎないことを認識しておくことが重要です。プロトコルの詳細を重視するあまり、ユーザ管理から監査、帯域幅スロットリング、脅威の検出まですべてを含む API 管理の全体像は見失いがちです。API はミッションクリティカルな企業アプリケーションへの直接的な道筋となることがよくあります。これはエンタープライズクラスのセキュリティ・ソリューションで保護する必要があります。

CA Technologies は、OAuth 対応エンタープライズ・アプリケーションへのインフラストラクチャの提供に力を注いでいます。CA Technologies は、既存の投資をアイデンティティおよびアクセス管理（IAM）テクノロジに完全に組み込み、エンタープライズ全体で一貫した認証モデルを実現するドロップイン・ソリューションを提供します。CA API Gateway ソリューションはすべて、導入の簡単な仮想イメージとして利用できます。また、CA Technologies は柔軟性が高く、現在の標準に完全に遵守しているとはかぎらないサードパーティの OAuth 導入と統合できます。これによって急速に発展する技術変化の影響を受けずに済みます。

CA Technologies に関するこのホワイト・ペーパーでは、OAuth とは何か、組織内の OAuth を簡単にする方法について説明します。

OAuth の簡単な例

ソーシャル・メディアは早くから OAuth を大々的に導入しました。Facebook と Twitter が大きな成功を収めたのは、それらが単なるスタンドアロンの Web サイトではなく、他のアプリケーションとの統合を促進するプラットフォームだったためです。その統合点は、個人の異なるアカウントの認証、認可、バインドの手段として通常 OAuth を使用する RESTful API です。

Twitter と Facebook は OAuth が実際に使用されている優れた事例です。多くの人はおそらく Twitter と Facebook の両方に別々のアカウントを持っているでしょう。アカウント名は似ていますが（セキュリティ上、パスワードは異なるはずです）、これらは異なるサイトが管理する別々のアカウントです。ではどのような設定を行えば、ツイートした内容がすぐに Facebook のウォールに表示されるようになるでしょうか。

以前であれば、おそらく Twitter のプロフィールに Facebook のユーザ名とパスワードを保存する必要があったでしょう。これによって新しくツイートするたびに、Twitter のアプリケーションが Facebook にログインしてクロスポストを行うことができます。この方法はパスワード・アンチパターンと呼ばれるようになりましたが、これは複数の理由でお勧めできません。Facebook のパスワードを Twitter に預けるということは、Twitter のアプリケーションに過大な権限を与えることを意味します。仮にハッカーがサイトに侵入したり内部の管理者が悪意ある行動をとった場合、プレーン・テキストのパスワードを利用して損害を及ぼすような写真を投稿したり、ユーザが Facebook に入れないようにしたり、アカウント自体を消去することすらありえます。

幸い、Twitter と Facebook では OAuth を使用してこの問題を解消しています。OAuth は Twitter が Facebook のウォールに投稿することのみを許可する認可委譲モデルを提供し、他のことは許可しません。これを以下の図 A に示します。

図 A

OAuth によって Facebook のパスワードを使用することなく、Twitter から自分の Facebook アカウントへの投稿を行えます。

ユーザの側から見ると、このインタラクションは非常に簡単で直観的です。以下の図 B は、その流れを示します。ユーザは Twitter の設定パネルから Facebook に移動するボタンをクリックするとそこでログインできます。これによって Facebook や Twitter のセキュリティ管理者が介入することなく、このユーザの 2 つの異なるアカウントが関連付けられます。ユーザは Facebook で認証されたら、ユーザの代わりに Twitter が操作を行えるよう許可したい特権のサブセットを選択できます。最後にユーザは自動的に Twitter に戻ってツイートを再開できます。このツイートは Facebook にも表示されるようになります。設定した Facebook と Twitter の連携は、ユーザが解除の操作を行うまで無期限に存続します。解除の操作ボタンは設定ページにあります。

図 B

Twitter が Facebook のウォールに投稿できるように認可する方法

ユーザにとってこれは簡単で直観的なプロセスで、それが OAuth の主たる魅力でもあります。しかし見えないところではサイト間でもっと複雑なやり取りが行われています。これは OAuth ダンスと呼ばれることがあります。3-legged OAuth はここで説明したシナリオのよく知られた名前です。これは OAuth 1.0a の仕様の最も典型的な使用事例で、現在は RFC 5849 として公開されています。

この仕様は詳細ですが驚くほど限定的です。これはユーザが自身のアカウントを関連付けて、限られた操作のサブセットを認可し、万能のパスワードではなく Twitter が安全にアクセスを継続できる不透明トークンを返せるようにするリダイレクション・フローを定義しています。また、少なくとも 1.0 バージョンではデジタル署名を使用してトークンをパラメータの内容にバインドする方法も詳細に記されています。これによって暗号化されない経路で送信されたコンテンツの整合性チェックを行うことができます。

OAuth 1.0a の仕様の強みの 1 つは、一般的な認可枠組みの定義というよりも、上記のような設計の共通課題への解決策の提供を目指していることです。これは問題を解決したいと考える人々による草の根の取り組みで、そのタイミングは完璧でした。当然ながら OAuth は広く成功を収め、Google、DropBox、SalesForce、FourSquare、LinkedIn などのサイトに実装されました。

しかし OAuth は進化しています。2012 年 10 月に公開された Version 2 は、もっと汎用性の高い一連の使用事例に対応することをめざしています。その結果、当然ソリューションは複雑になり、OAuth を実装して企業の API を保護しようとする開発者にとっては困難が増しています。

OAuth 以外の取り組み

OAuth が対応する認可委譲の問題を解決するための十分に定義された明確なプロセスはありません。OAuth の設計者は代替案を考慮し、ほんの少数の（完全に専用の）解決策を考えつきました。必要は明らかに OAuth の発明の母でしたが、主な目的はオープン性でした。

連携シングル・サインオン（SSO）によく使用される SAML が、Sender-Vouches トークン・タイプを使用してサイト間の委任操作を通信するトークン形式として使用できるという考えは、確かにありえます。しかし SAML 自体は信頼関係またはアカウントのバインドを設定するインタラクションのフローを定義することはできません。また、SAML は非常に複雑な XML 形式であるため、RESTful の理念やシンプルな JSON データ構造を中心とする現在の開発プラクティスにはなじみません。

OpenID Connect は、Web でシングル･サインオンを提供しようと試みました。OpenID Connect が広く行き渡った理想的な世界なら、OAuth は必要ないかもしれません。しかし、実際には、Yahoo や WordPress などの影響力のあるサイトで成功を収めているとはいえ、OpenID Connect の普及はまったく進んでいません。それでも、OpenID Connect は OAuth をうまく補完できることから、今後もチャンスがあるでしょう。

OAuth 2.0 と旧バージョンの違い

OAuth 1 は需要によって急速に進化しました。共通する問題に解決策を提供し、主要なソーシャル・メディアのアプリケーションによって導入されたことで広く普及しました。現在最も一般的に実装されているのは 1.0a で、これは最初の仕様に少々変更が加えられ、セキュリティの脆弱性に対応しています。

1.0a の仕様は設計に優れ完成度も高いものですが、使用事例は限定的です。これは強みの 1 つといえます。1 つのことを実行し、それを非常にうまく実行できるのです。OAuth 1.0 は幅広くサポートされ、ほとんどの言語でライブラリを使用できます。しかしいまだに手作業のイメージがあるところが個人の開発者にとっては魅力ですが、企業には敬遠されています。

OAuth 1.0a はより複雑なため、幅広い導入には至っていません。特に、JavaScript などの言語を使用してコーディングするには困難な暗号化プロセスなど、複雑な処理を強いられます。たとえば OAuth 1.0a ではクライアントは HTTP パラメータに署名する必要があります。これは暗号化されてない転送（従来の Web 上で一般的な使用パターン）には有効ですが、API にはそれほど有効ではありません。

パラメータを正規化する必要があるため（たとえば命令の正規化、エスケープ文字列の処理など）、署名プロセスそのものが複雑です。署名を適用する方法についての解釈はクライアントとリソースのサーバで異なることが多いため、これは開発者にとって大きなストレスの要因でした。

ここで役立つライブラリは確かにありますが、もっと問題なのは署名の要件のために、開発者が cURL などの簡単なコマンドライン・ユーティリティを使用してトランザクションをテストする機能が制限されることです。SOAP Web サービスなどの代替案と比べて RESTful スタイルが魅力的なのは、コーディングに特別なツールが必要ないからです。OAuth の署名作業はこのメリットを損ねてしまいます。

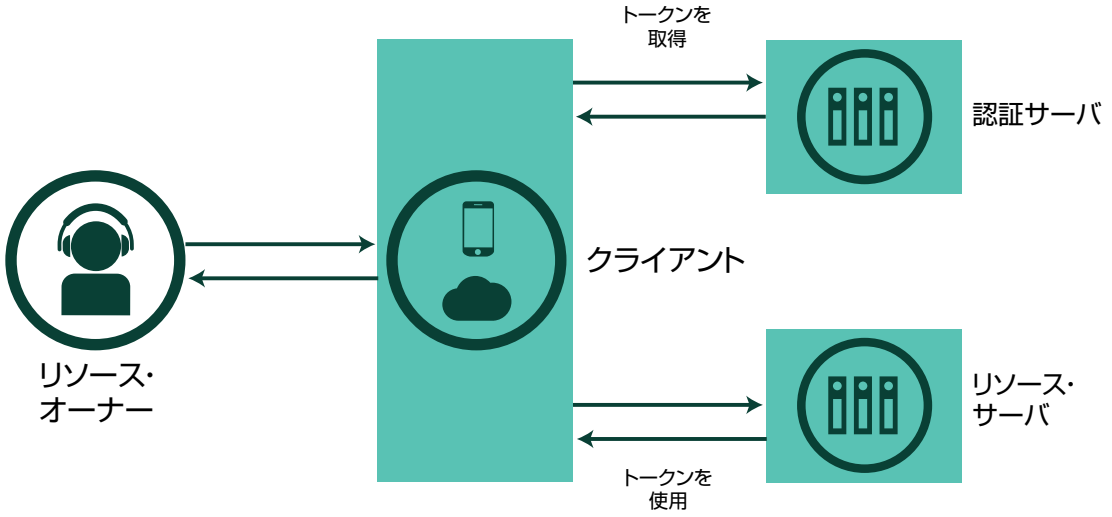
初期の仕様でもクライアントのタイプに対する視点が限られていました。3-legged の場合でも、クライアントは通常、Web アプリケーションでした。しかし開発者はブラウザ内で実行しているアプリケーションや、携帯電話やタブレットなどのモバイル・デバイス上で実行しているスタンドアロンのアプリケーションで OAuth を使用したいとますます望むようになっています。これは OAuth 1.0 では可能ですが、ブラウザとアプリケーションの間でコピー・アンド・ペーストを行わなければならず不便で、ユーザ・エクスペリエンスは不十分です。

OAuth 2.0 はクライアント開発を簡略化し、全体的なユーザ・エクスペリエンスを改善し、OAuth の導入の規模を変更するために、オリジナルの OAuth の実装を一般化することを試みています。これには大幅な変更が必要で、以前のバージョンとの後方互換性はありません。

新しい仕様は認可の役割をアクセス制御から明確に分離しています。現在、認可サーバはリソース・サーバからはっきり分離されています。また、これは役割の論理的分離とは別に、従来の SSO アーキテクチャのように、一元的な認可サーバの使用と、複数のリソース・サーバの分散を促進します。OAuth 2.0 の認可サーバは RESTful のセキュリティ・トークン・サービス（STS）に相当します。

図 C

OAuth 2.0 では認可サーバとリソース・サーバを明確に分け、さらにトークンの取得について示すフローを定義します。



OAuth 2.0 は、従来の Web アプリケーション、ユーザ・エージェント（Web ブラウザ）内ベースのアプリケーション、ネイティブ・アプリケーション（モバイル電話アプリケーション、セットトップ・ボックス、ゲーム・コンソールなど）という 3 つのクライアント・プロフィールのサポートを試みています。これらはそれぞれリソース・オーナー、認可サーバ、保護されたリソースの間のインタラクションの点で、異なる機能を有しています。また、それぞれが異なるセキュリティ用件に従う必要がある場合があります。この仕様書では、こうした多様なニーズに対応するための複数の新しい認可グラントについて記載されています。このグラントは、クライアントがリソースへのアクセスの認可を取得できるプロセスについて説明しています。

グラントには以下のものがあります。

- **認可コード** — このグラントはクライアントが Twitter などの Web アプリケーションである、典型的な 3-legged シナリオを表します。中間的な認可コードを使用し、リソース・オーナーのユーザ・エージェント（ブラウザ）を使用して認可サーバからクライアントに安全に認可を委譲します。クライアントにリソース・オーナーのクレンデンシャルが共有されることはなく、乗っ取られる可能性があるリソース・オーナーのユーザ・エージェントにアクセス・トークンが共有されることはないというメリットがあります。

- **インプリシット** — これは JavaScript アプリケーションなどユーザ・エージェント内で実行するアプリケーションに最適な、もう少し単純なグラントです。クライアントは認可サーバから直接アクセス・トークンを取得します。これによって仲介の認可コードの複雑さの多くが解消されますが、リソース・オーナーがアクセス・トークンを取得できる可能性があるという欠点があります。
- **リソース・オーナー・パスワード・クレデンシャル** — このグラントではリソース・オーナーが直接クレデンシャルをクライアントと共有し、クライアントはこれを使用して、1 回のトランザクションでアクセス・トークンを直接取得します。クライアントはアクセス・トークンを保護対象のリソースとのその後のすべてのインタラクションに使用するため、クレデンシャルは存続できません。これは非常に単純なフローですが、リソース・オーナーとクライアントの間に信頼が必要です。
- **クライアント・クレデンシャル** — このフローではクライアントは自身のクレデンシャルを使用してリソースにアクセスします。そうすることでクライアントの既存の権限を活用します。

これらのグラントに加えて、他の形態の認可に対応する拡張メカニズムもあります。たとえば、OAuth アクセス・トークンを取得する手段としての SAML トークン使用について説明する SAML ベアラー・トークンという仕様があります。これは従来のブラウザ SSO インフラストラクチャと最新の API の間のブリッジとなるため非常に重要です。

OAuth 2.0 ではセッション管理をより効果的にサポートできるようにするため、トークンが変更されました。以前はアクセス・トークンは長期的に存続でき（最大 1 年）、Twitter のように存続期間が無制限というものもありました。OAuth 2.0 は存続期間の短いトークンと長期的な認可という概念を導入しました。認可サーバは現在、クライアント・リフレッシュ・トークンを発行します。これは長期的なトークンで、クライアントが短期間有効なアクセス・トークンを取得するために、複数回使用することができます。そのメリットの 1 つは、クライアントが必要に応じて新しいトークンを取得することできないように、リソース・オーナーまたはセキュリティ管理者のいずれかが容易に遮断できることです。

トークン署名は新しいオプションです。優先されるのは、SSL で保護された単純なベアラー・トークン（トークンを直接使用してアクセス権を取得し、機密が保護されると考えられる）を使用することです。これは署名の処理よりずっと簡単ですが、その後も単純な形態で存在して SSL 以外のトランザクションをサポートします。

OAuth が難しい理由

OAuth の単純な概念実証の構築は難しくありません。大多数の主要言語で書かれたライブラリは、手動のコーディングとエンドツーエンドの OAuth の実証における課題に役立ちます。しかし OAuth の大規模な実装については、トランザクションのボリューム、保護すべき API の数、多様なクライアントの数などがすべて規模拡大の要因となり、実装や運用を担当するグループにとっては依然として大きな課題です。

OAuth 2.0 は移動する標的のようなものです。1.0a の仕様は 1 つの問題を効果的に解決しました。しかし新しい仕様では適用範囲と一般化を強化したことで、相互運用性をきわめて困難にするあいまいさがもたらされました。そのために、OAuth の動向の中心的な構成要素であるソーシャル・ネットワーキング・アプリケーションの多くのが 1.0 の仕様のままで、事態が収まるまで様子を見ています。

トークンの形式の公開はこれをよく表しています。これは一方では、初期の仕様で開発者にとって困難だった署名プロセスを大幅に簡略化し、また、様々なトークン（SAML など）をカプセル化できる機能をもたらし、既存の投資を活用する機会を開きましたが、相互運用性の重大な課題ももたらします。

OAuth に関して現在よくある間違いは、OAuth を単独でとらえがちなことです。OAuth は実際に魅力的なトレンドですが、企業のアクセス制御の課題のほんの一部にすぎません。認可はクライアントによってすべて決定されるわけではなく、保護対象のリソースをホストする企業も制御の手段を持つ必要があります。単一の OAuth の実装はこの相互関係をほとんど承認しませんが、相互の信頼と制御は企業に取って不可欠です。

OAuth は、単なるスタンドアロンのソリューションではなく、企業の API にとって一般的なポリシーベースのアクセス制御システムの一部を形成する必要があります。ポリシーベースのアクセス制御では、両者がアクセスの制御を行えます。時刻の制限や IP のホワイトリスト / ブラックリストなどの制御を組み入れています。SQL インジェクションやクロスサイト･スクリプティング（XSS）攻撃などの脅威を識別し無効化します。パラメータとメッセージ･コンテンツ（JSON または XML を含む）を受入可能な値と比較して検証します。また、企業の監査システムと完全に統合できるため、誰がいつ何にアクセスしたかをリソース提供者は正確に把握できます。最後に、豊富なポリシーベースのアクセス制御によって、ネットワーク通信を形成し、トランザクションを利用可能なサーバにルーティングし、過剰なトラフィックがユーザ・エクスペリエンスやサーバに影響を及ぼす前に調整することで、SLA の管理を可能にします。

企業は自社の Web サイト用の IAM インフラストラクチャを構築することは考えないでしょう。アーキテクトや開発者はこれには単純な HTTP の基本認証よりも多くの作業があると認識しています。OAuth も同様です。一見簡単そうに見えますが、最終的にはきわめて複雑な全体的な認可プロセスの 1 つです。

CA API Gateway は OAuth の実装にどのように役立つか

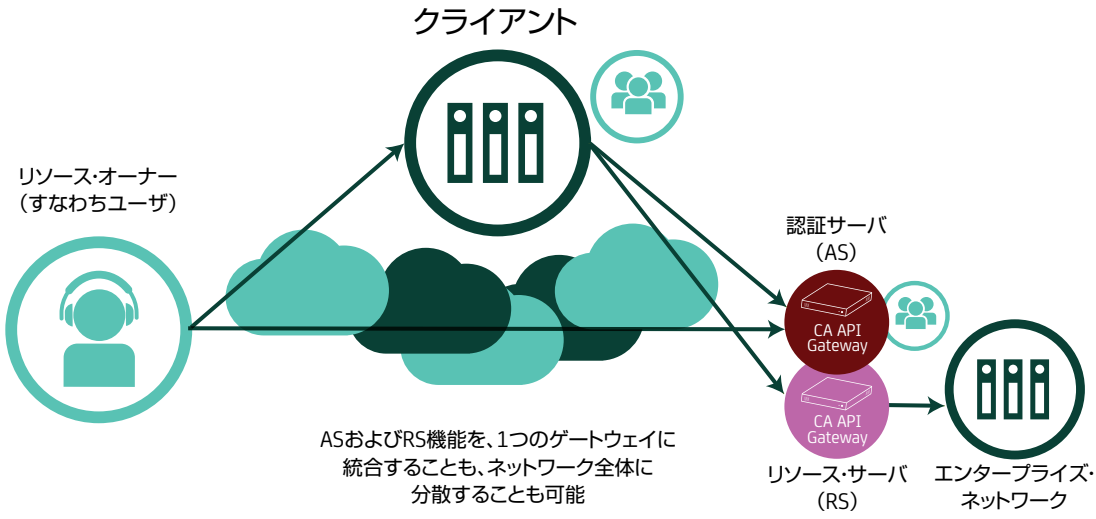
CA Technologies は、OAuth 1.0a と OAuth 2.0 の実装のための完全なターンキー･ソリューションを提供します。OA uth は、CA API Gateway の豊富なポリシー・ベースのアクセス制御エンジンに含まれています。これは 1 つのゲートウェイ・インスタンスで 1 秒あたり数万のトランザクションを処理する、真に大規模な OAuth の使用です。これらのゲートウェイはハードウェア・アプライアンスまたは低価格の仮想イメージとして導入できます。どちらのフォーム・ファクタも企業の OAuth に軍事レベルのセキュリティ・インフラストラクチャをもたらし、1 つのパッケージに FIPS 認定の暗号モジュール、高度な脅威検出、SLA トラフィック管理、きめ細かいアクセス制御が組み込まれています。鍵だけでなく、ドアの前に護衛がいるようなものです。

CA Technologies の API ゲートウェイは、認証サーバ（AS）としても、保護リソース・サーバ（RS）としても導入可能です。どちらの構造要素でも、単一のゲートウェイ・インスタンスに統合したり、分離することができ、図 D に示すように、集中 A から多数の分散した RS インスタンスにサービスを提供することができます。

CA API Gateway は、ハードウェアと仮想アプライアンス・フォーム・ファクタの形式があるため、非常に幅広い導入に対応します。ハードウェア・ゲートウェイはオンボード・ハードウェア・セキュリティ・モジュール（HSM）で使用可能で、ほとんどの安全な環境に主要な保護を提供します。仮想アプライアンスによって導入が簡単になり、デスクトップ･コンピュータから最もパワフルなサーバ･インフラストラクチャまでどこでも実行することができます。

図 D

CA Technologies の API ゲートウェイは、OAuth 実装を容易にします。



OAuth Toolkit のメリット

CA API Gateqay の OAuth Toolkit は、通常の OAuth 導入向けの、すぐに動作できるよう設計された標準テンプレートを使用します。これを使用することで、数日間ではなく数分間で顧客は堅牢な OAuth Toolkit 機能を既存の API に追加できます。

実際のところ、多数のベンダが約束する万能サイズのソリューションは、一部の制約のない環境以外ではうまく機能することがほとんどありません。たとえば、現実の大多数のプロジェクトには、統合する PKI インフラストラクチャにアクセスするために必要なアイデンティティ・システムがすでに存在します。業界全体でアプリケーションとデータの統合はうまくいっていますが、セキュリティの統合は依然として現在進行中の課題です。

こうした統合の問題により効果的に対応するために、CA API Gateqay は暗号化、パラメータの正規化、セッション管理など基本的な OAuth コンポーネントを提供します。これらは当社の完全なターンキー・ソリューションで使用しているのと同じ基本的なコンポーネントですが、アクセス制御ポリシー内で完全に設定可能なアサーションとして発表されました。これによってアーキテクトや開発者は OAuth Toolkit の実装を調整して、直面する可能性があるほぼすべての問題に対応することができます。

OAuth Toolkit の承認手続きのカスタマイズも、ゲートウェイ・テンプレートのオープン性から大きなメリットが得られる領域で、柔軟でオープンなツールキットによって強化されています。最初の信頼の設定は、OAuth Toolkit の全体的なプロセスにとってきわめて重要な部分です。CA API Gateqay では、この手順を完全にカスタマイズして、既存のアイデンティティ・インフラストラクチャと統合させ、企業のコンプライアンス要求に対応しています。

CA Technologies は 2-legged OAuth または 3-legged OAuth の使用事例にどのように役立つか

CA API Gateqay は、保護サービスにエンドポイントの認証サービスとアクセス制御の両方を提供できます。これら 2 つの機能は単一のゲートウェイに共存することも、分離することも可能です。分離することのメリットはスケーラビリティ、冗長性、サービスの地理的分散に関することにあります。また、企業と公共の API の物理的なパーティショニングなど、ビジネス・ケースとの整合も可能です。大半の企業は保護すべき API を多数保有しており、これらは複数の場所から提供されます。こうした場合、CA Technologies の集中 API ゲートウェイを認証サーバとして導入し（冗長を目的としてクラスタで導入）、ゲートウェイのリモート・クラスタで特定の API インスタンスを保護することは理にかなっています。

これらの導入パターンではどちらも OAuth 1.0a および 2.0 バージョンを同時に使用可能です。また、このパターンは従来の 2-legged および 3-legged のシナリオの両方と、SAML ベアラー・トークンなどの拡張グラントを含む OAuth 2.0 グラント・モデルに対しても有効です。図 E と図 F は、これらの導入について示します。

図 E

従来の 2-legged のシナリオや、リソース・オーナー・クレデンシャルやクライアント・クレデンシャルなどのグラントの通常の導入

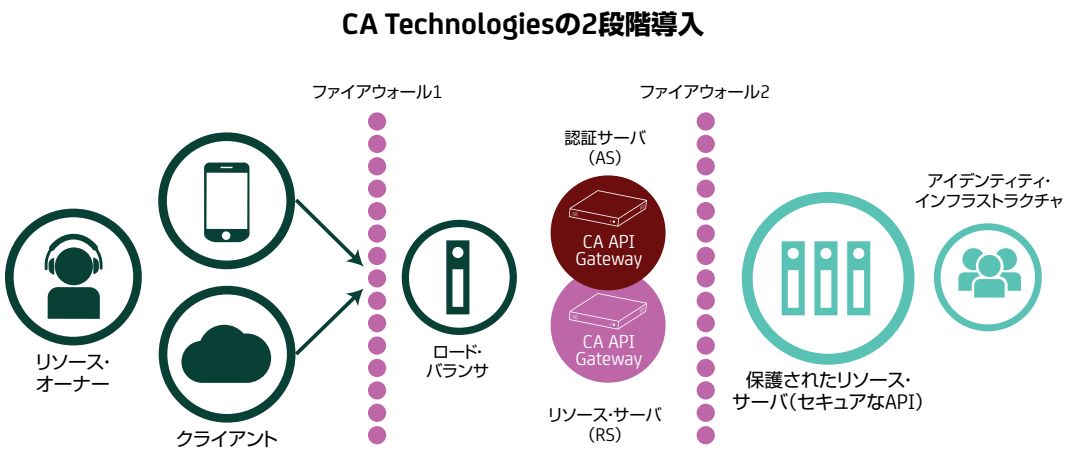
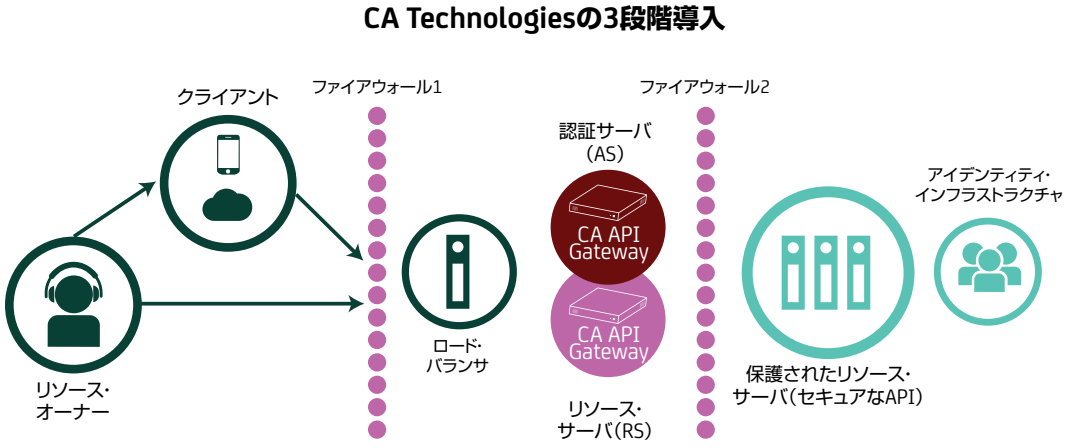


図 F

通常の 3-legged の導入シナリオと認可コード・グラントおよびインプリシット・グラント・タイプ CA API Gateway は、同時にすべての OAuth バージョンとカスタム・マッピングをサポートでき、相互運用性の課題に対応します。



CA Technologies の連絡先

CA Technologies は、質問、コメントー般的なフィードバックを受けたまわっています。詳細については、CA Technologies の担当者にご連絡いただくか、**www.ca.com/jp/api** をご覧ください

ca.com/jp/でCA Technologiesにアクセスしてください

CA Technologies（NASDAQ:CA）は、企業の変革を推進するソフトウェアを作成し、アプリケーション・エコノミーにおいて企業がビジネス・チャンスを獲得できるよう支援します。ソフトウェアはあらゆる業界であらゆるビジネスの中核を担っています。プランニングから開発、管理、セキュリティまで、CA は世界中の企業と協力し、モバイル、プライベート・クラウドやパブリック・クラウド、分散環境、メインフレーム環境にわたって、人々の生活やビジネス、コミュニケーションの方法に変化をもたらしています。詳細については **ca.com/jp** をご覧ください。