WHITE PAPER • MAKE OAUTH IMPLEMENTATION SIMPLE FOR YOUR ORGANIZATION

A How-to Guide to OAuth & API Security

Table of Contents

What is OAuth?

OAuth is an emerging Web standard for authorizing limited access to applications and data. It is designed so that users can grant restricted access to resources they own—such as pictures residing on a site like Flickr or SmugMug—to a third-party client like a photo printing site. In the past, it was common to ask the user to share their username and password with the client, a deceptively simple request masking unacceptable security risk. In contrast to this, OAuth promotes a least privilege model, allowing a user to grant limited access to their applications and data by issuing a token with limited capability.

OAuth is important because it places the management of Web delegation into the hands of the actual resource owner. The user connects the dots between their accounts on different Web applications without direct involvement from the security administrators on each respective site. This relationship can be long-lasting but can easily be terminated at any time by the user. One of the great advancements OAuth brings to the Web community is formalizing the process of delegating identity mapping to users.

OAuth hs become a foundation standard of the modern Web and has grown far beyond its social media roots. OAuth is now very important for the enterprise; insurance companies, cable operators and even healthcare providers are using OAuth to manage access to their resources. Much of this adoption is driven by the corporate need to support increasingly diverse clients and mobile devices, in particular. These organizations are aggressively deploying APIs to service this new delivery channel and OAuth is the best practice for API authorization.

But it is important to recognize that OAuth is only one component of a full API access control and security solution. It is easy to focus on the details of the protocol and lose sight of the big picture of API Management—encompassing everything from user management to auditing, throttling and threat detection. APIs often represent a direct conduit to mission-critical enterprise applications. They need an enterprise-class security solution to protect them.

Layer7 is committed to providing infrastructure to OAuth-enable enterprise applications. We offer drop-in solutions that fully integrate with existing investments in identity and access management (IAM) technology to provide a consistent authorization model across the enterprise. All Layer7 API Gateway solutions are available as simple-to-deploy virtual images. Layer7 also provides the flexibility to integrate with third-party OAuth implementations that may not be entirely compliant with the current standards, thus insulating you from the changes that come from a rapidly-evolving technology.

This white paper from Layer7 describes what OAuth is and shows how you can make OAuth simple in your organization.

Can You Provide a Simple OAuth Example?

Social media has been the largest early adopter of OAuth. Facebook and Twitter owe much of their success to the fact that they are not simply standalone Web sites but platforms that encourage integration with other applications. The integration points are RESTful APIs that typically use OAuth as a means of authentication, authorization and binding together of different personal accounts.

Twitter and Facebook provide excellent examples of OAuth in action. Like many people, you probably have separate accounts on both of these social media powerhouses. Your account names may be similar (and in the name of good security, hopefully you use different passwords) but they are distinct accounts managed on different sites. So, how can you set things up so that your tweets show up instantly on your Facebook wall?

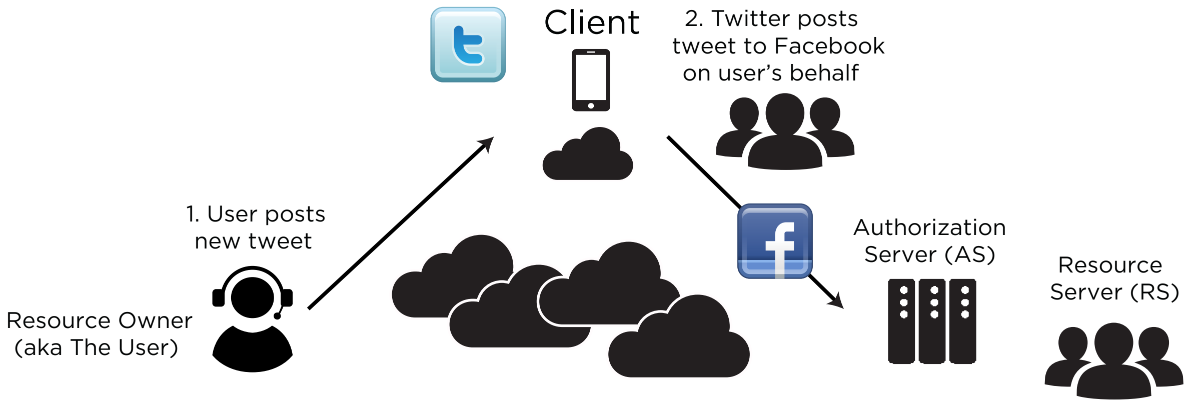
In the past, you would probably have had to store your Facebook username and password in your Twitter profile. This way, whenever you published a new tweet, the Twitter application could sign on for you to cross-post it onto Facebook. This approach has come to be called the password anti-pattern and it is a bad idea for a number of reasons. Entrusting Twitter with your Facebook password simply gives this application too much power. If a hacker was to compromise the site or an internal administrator went rogue, they could leverage your plain text password to post damaging pictures, lock you out of Facebook or even delete your entire account.

Fortunately, Twitter and Facebook both use OAuth to overcome this challenge. OAuth provides a delegated authorization model permitting Twitter to post on your wall—but nothing else. This is shown in Figure A below.

Figure A:

OAuth allows Twitter to post tweets to your Facebook account without using your Facebook password.

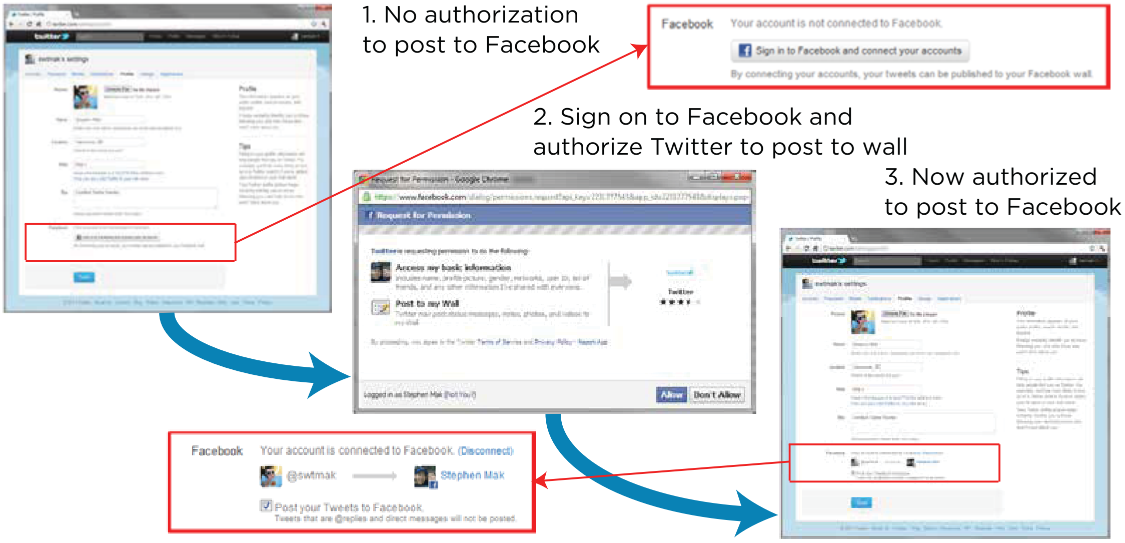
Letting Twitter Post Tweets to Your Facebook Wall


Client
2. Twitter posts tweet to Facebook on user's behalf
1. User posts new tweet
Resource Owner (aka The User)
Authorization Server (AS)
Resource Server (RS)

From the user perspective, the interaction is very simple and intuitive. You can follow it in Figure B below. From their Twitter settings panel, a user clicks on a button that transfers them to Facebook, where they can sign in. This creates an association between this user's two separate accounts without any involvement from Facebook or Twitter security administrators. Once authenticated on Facebook, the user undergoes a consent ceremony, where they can choose the subset of privileges they want to grant to Twitter to permit the application to perform actions on their behalf. Finally, the user returns automatically to Twitter, where they can resume posting tweets, which now appear on their Facebook wall as well. The relationship they have set up persists indefinitely or until they decide to break it explicitly, using controls found on the settings page.

Figure B:

How a user authorizes Twitter to post tweets on their Facebook wall.



1. No authorization to post to Facebook

Facebook Your account is not connected to Facebook.

Sign in to Facebook and connect your accounts

By connecting your accounts, your tweets can be published to your Facebook wall.

2. Sign on to Facebook and authorize Twitter to post to wall

3. Now authorized to post to Facebook

Facebook Your account is connected to Facebook. (Disconnect)

@swtmak Stephen Mak

☑ Post your Tweets to Facebook.
Tweets that are @replies and direct messages will not be posted.

For the user, this is a simple and intuitive process—and indeed, that is much of OAuth's appeal. But underneath the hood is a much more complex interaction between the sites, often called the OAuth dance. Three-legged OAuth is the popular name for the scenario described here; it is the most typical use case for the OAuth 1.0a specification, now published as RFC 5849.

This specification is detailed but surprisingly narrow. It defines the redirection flow that allows a user to associate their accounts, to authorize a limited subset of operations and return an opaque token that Twitter can persist safely for access instead of an all-powerful password. It even details—at least in the 1.0 version—a method for binding the token to parameter content using digital signatures, thus allowing integrity checks on content submitted over unencrypted channels.

One of the strengths of the OAuth 1.0a specification is that, rather than attempting to define a generalized authorization framework, it instead set out to offer a solution to the common design challenge described above. It was a grass-roots initiative by people with a problem to solve and its timing was perfect. Unsurprisingly, it became wildly successful, seeing implementation on sites such as Google, DropBox, SalesForce, FourSquare and LinkedIn.

OAuth, however, has evolved. Version 2, which was published in October 2012, ambitiously aims to satisfy a much more generalized set of use cases. This naturally adds complexity to the solution and adds to the difficulty faced by developers trying to implement it to protect enterprise APIs.

Hasn't This Problem Been Solved Before?

There are not any fully defined, formal processes for solving the delegated authorization problem addressed by OAuth. Its designers considered the alternatives and came up with only a handful of (completely proprietary) solutions. Necessity was certainly the mother of OAuth's invention but openness was a key goal.

It is certainly conceivable that SAML, most often used for federated Single Sign-On (SSO), could be used as a token format to communicate delegated operations between sites using the sender-vouches token type. However, SAML on its own does not define the flow of interactions to set up the trust relationship or account bindings. Furthermore, as a very complex XML format, SAML does not sit well with current development practices focused on RESTful principles and simple JSON data structures.

OpenID Connect attempted to offer a single Web sign-on. In a perfect world where OpenID Connect was universal, then OAuth might never have been necessary. But despite success at influential sites such as Yahoo and WordPress, OpenID Connect has never seen widespread adoption. Nevertheless, OpenID Connect may have a second chance for success because of how it complements OAuth.

How Does the OAuth 2.0 Differ from Previous Versions?

OAuth 1 evolved very rapidly because of demand; it offered a solution to a common problem and its adoption by leading social applications gave it wide exposure. The most common implementation at present is 1.0a, which incorporates a slight modification from the original specification, to address a security vulnerability.

The 1.0a specification is well-designed and quite complete but only for a narrow set of use cases. Arguably, this is one of its strengths; it does one thing, and does it very well. OAuth 1.0 enjoys wide support, with libraries available in most languages. Still, it suffers from a largely do-it-yourself feel—a characteristic that may appeal to individual developers but one that leaves the enterprise cold.

OAuth 1.0a had some additional complexities that have held it back from enjoying wider acceptance. It pushes complexity into clients—particularly around cryptographic processing—which can be difficult to code using languages like JavaScript. For example, OAuth 1.0a requires that clients sign HTTP parameters; this is a good idea for unencrypted transmissions (a common usage pattern on the conventional Web) but less so with APIs.

The signing process itself is confusing because of the need to canonicalize parameters (for example, normalizing ordering, dealing with escape sequences etc.) This has been a major source of developer frustration, as clients and resource servers often have different interpretations of how signatures are applied.

Certainly there exist libraries that can help here, but a bigger problem is that the signature requirement limits the ability of developers to test transactions using simple command-line utilities like cURL. Much of the appeal of the RESTful style over using alternatives like SOAP Web services is that the former requires no special tools to code. OAuth signatures work against this advantage.

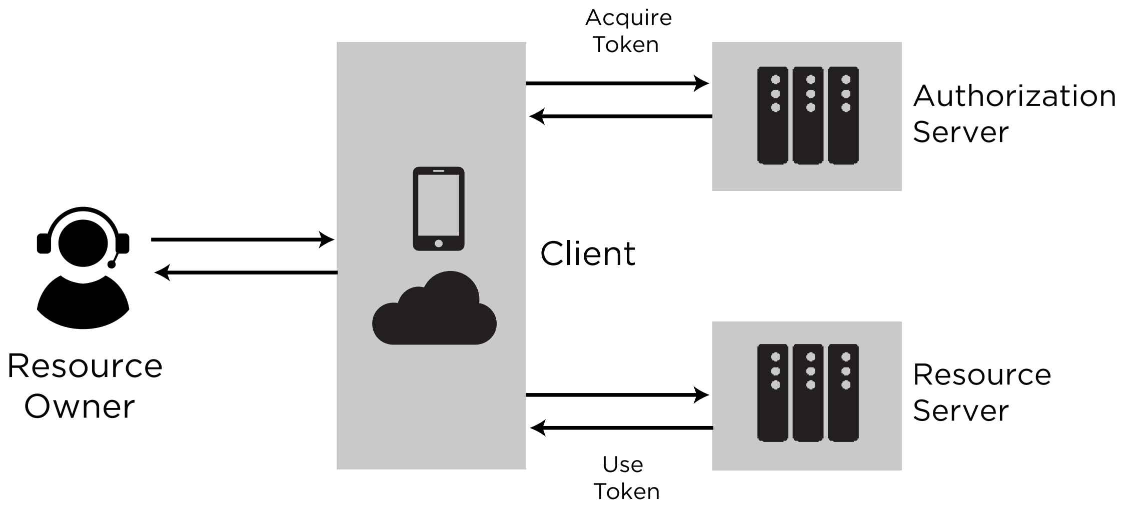
The earlier specification also had a limited view of client types. In the cleanest, three-legged case, the client was usually a Web application. However, developers increasingly wanted to use OAuth in applications running inside a browser or within standalone apps running on mobile devices like phones or tablets. This is possible with OAuth 1.0 but the user experience is poor, as it may involve an awkward copy-and-paste operation between a browser and the app.

OAuth 2.0 generalizes the original OAuth implementation in order to simplify client development, improve the overall user experience and scale OAuth deployments. This required significant changes, not backwardly compatible with previous versions.

This specification explicitly separates out the roles of authorization from access control. Now the authorization server is cleanly separated from the resource server. Aside from the logical segregation of roles, this also promotes use of a central authorization server and distribution of multiple resource servers, much like a classical SSO architecture. In fact, an OAuth 2.0 authorization server is really the RESTful equivalent of a security token service (STS).

Figure C:

OAuth 2.0 makes a clear distinction between Authorization server and resource server and further defines the flows describing token acquisition.



OAuth 2.0 attempts to support three client profiles: conventional Web applications; applications based inside a user-agent (that is, a Web browser); native applications (such as a mobile phone app, a set-top box or even a game console). Each of these may have different capabilities in terms of interacting between resource owners, authorization servers and protected resources. Each may also be subject to different security requirements. The specification describes a number of new authorization grants to accommodate these diverse needs. The grants describe a process by which a client can acquire authorized access to a resource.

These grants include:

- **Authorization Code** — This grant describes the typical three-legged scenario, where the client is a Web application such as Twitter. It uses an intermediate authorization code to securely delegate authorization from an authorization server to a client via the resource owner's user agent (browser). It has the benefits that a resource owner's credentials are never shared with the client, nor is the access token ever shared with the resource owner's user agent where it could be hijacked.

- **Implicit** — This is a slightly simpler grant that is best suited to applications running inside a user agent, such as JavaScript apps. The client directly acquires an access token from the authorization server. This eliminates much of the complexity of the intermediary authorization code but has the drawback that the resource owner could potentially get the access token.
- **Resource Owner Password Credentials** — In this grant, the resource owner shares credentials directly with the client—which uses these to obtain an access token directly, in a single transaction. The credentials are not persisted, as the client uses the access token for all subsequent interactions with protected resources. This is a very simple flow but it demands that there be trust between a resource owner and a client.
- **Client Credentials** — In this flow, the client uses its own credentials to access a resource. Thus, it is really leveraging the client's existing entitlements.

In addition to these grants, there is an extensibility mechanism to accommodate other forms of authorization. For example, a SAML bearer token specification exists that describes the use of SAML tokens as a means of acquiring OAuth access tokens. This is very important because it represents a bridge between classic browser SSO infrastructure and modern APIs.

Tokens have changed in OAuth 2.0, to better support session management. In the past, access tokens were very long-lived (up to a year) or—as is the case with Twitter—they had an unlimited lifespan. OAuth 2.0 introduces the concept of short-lived tokens and long-lived authorizations. Authorization servers now issue client refresh tokens. These are long-lived tokens a client can user multiple times to acquire short-term access tokens. One of the benefits of this is that either resource owners or security administrators can easily cut off clients from acquiring new tokens if they need to.

Token signatures are now optional. The preference is to use simple bearer tokens (meaning the token is used directly to gain access and is considered secret) protected by SSL. This is much simpler than processing signatures, though the later still exist in a simplified form to support non-SSL transactions.

Why is OAuth Hard to Do?

Building a simple OAuth proof-of-concept is not difficult; there are libraries in most major languages that can help with the challenge of hand coding an end-to-end OAuth demonstration. However, implementing OAuth at scale—where transaction volume, the number of APIs to protect and the number of diverse clients all contribute to scale—remains a great challenge for any development and operations group.

OAuth 2.0 is also a moving target. The 1.0a specification solved one problem and solved it well. But the increased scope and generalization of the new specification has created considerable ambiguity that makes interoperability extremely challenging. This is why many social networking applications—the core constituency of the OAuth movement—remain at the 1.0 spec, waiting for things to settle down.

The opening of token formats illustrates this nicely. While, on the one hand, this has greatly simplified the signature process, which was challenging for developers in the earlier specifications, it has also introduced the ability to encapsulate different tokens (such as SAML)—opening up opportunities to leverage existing investments but also creating significant interoperability challenges.

The biggest mistake people make with OAuth today is looking at it in isolation. OAuth is indeed a compelling trend but it is only one piece of the enterprise access control puzzle. Authorization cannot be dictated entirely by the client; the enterprise hosting the protected resource must also have control. Single-point OAuth implementations rarely acknowledge this two-way street but reciprocal trust and control are essential for the enterprise.

OAuth must be a part of the general policy-based access control system for enterprise APIs, not simply a standalone solution. Policy-based access control gives both parties control over access. It incorporates controls such as time-of-day restrictions and IP white/black lists. It identifies and neutralizes threats like SQL Injection or Cross-Site Scripting (XSS) attacks. It validates parameter and message content (including JSON or XML) against acceptable values. It integrates fully with enterprise audit systems so resource providers know exactly who is accessing what and when. And finally, rich policy-based access control allows management of SLAs by shaping network communications, routing transactions to available servers and throttling excess traffic before it can affect user experience or threaten servers. The enterprise would never consider building its own IAM infrastructure for its Web site—architects and developers recognize that there is much more to this than simple HTTP basic authentication. OAuth is similar—deceptively simple but ultimately a piece of a very complex overall authorization process.

How Does the Layer7 API Management Help Me Implement OAuth?

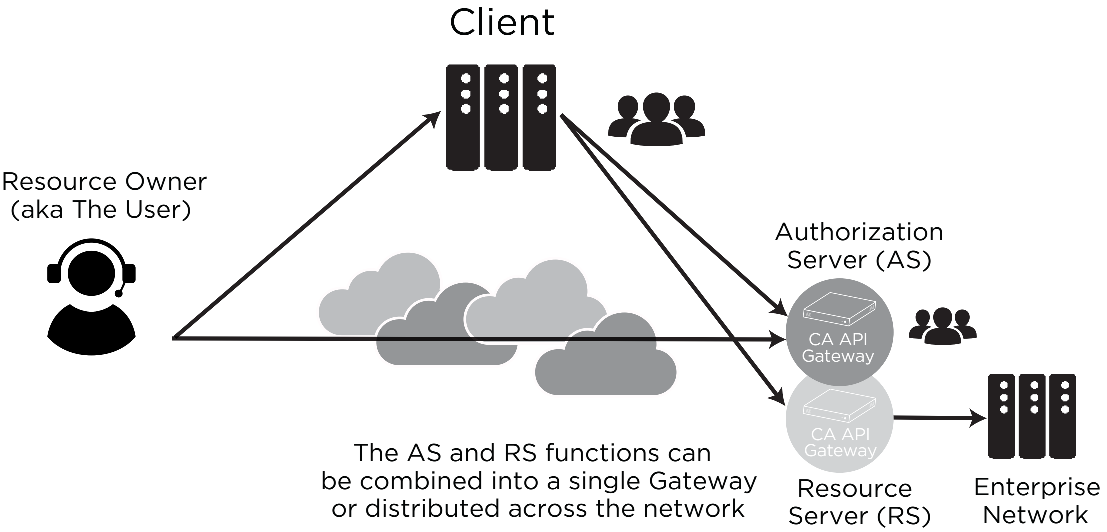
Layer7 provides a complete, turnkey solution for OAuth 1.0a and OAuth 2.0 implementations. OAuth is included within the rich, policy-based access control engine in Layer7 API Management. This is truly OAuth at scale, handling tens of thousands of transactions per second on a single Gateway instance. These Gateways can be deployed as hardware appliances or low-cost virtualized images. Both form factors bring military-grade security infrastructure to enterprise OAuth, incorporating FIPS-certified cryptographic modules, advanced threat detection, SLA traffic management and fine-grained access control in a single package. It is like having a guard at your door, instead of just a lock.

Layer7 API Management can be deployed both as authorization servers (AS) and protected resource servers (RS). Both architectural elements can be merged into a single Gateway instance or they can be separated, allowing a centralized AS to service many distributed RS instances, as illustrated in Figure D.

Because Layer7 API Management come in both hardware and virtual appliance form factors, they support the widest possible range of deployments. Hardware Gateways are available with onboard hardware security modules (HSMs), providing key protection for the most secure environments. Virtual appliances make deployment simple and can run anywhere from the desktop to the most powerful server infrastructure.

Figure D:

Layer7 API Management makes implementing OAuth simple.


Client
Resource Owner (aka The User)
Authorization Server (AS)
CA API Gateway
CA API Gateway
The AS and RS functions can be combined into a single Gateway or distributed across the network
Resource Server (RS)
Enterprise Network

What is the Benefit of an OAuth Toolkit?

The Layer7 AP:I Management OAuth Toolkit uses standardized templates designed to work out-of-the box for typical OAuth Toolkit deployments. Using these, customers can add robust OAuth Toolkit capabilities to existing APIs in minutes, instead of days.

The truth is, however, that the one-size-fits-all solution promised by so many vendors rarely works well outside of a very limited, green-field opportunity. For example, most real-world projects have existing identity systems they need to access or PKI infrastructure they need to integrate with. As an industry, we are very good at application and data integration; security integration remains an ongoing challenge.

To better address these integration challenges, the Layer7 API Management also provides basic OAuth Toolkit components, from cryptography to parameter canonicalization to session management. These are the same basic components used in our complete, turnkey solution but surfaced as completely configurable assertions within an access control policy. This allows architects and developers to tune their OAuth Toolkit implementations to meet nearly any challenge they may face.

Customizing the OAuth Toolkit consent ceremony is another area that greatly benefits from the openness of the gateways templates, augmented by the power of a flexible and open toolkit. Setting up initial trust is a critical part of the entire OAuth Toolkit process. Layer7 API Management allows you to fully customize this step to ensure that it integrates with existing identity infrastructure and meets enterprise compliance demands.

The OAuth Toollkit includes OpenID Connect, the industry standard authentication protocol that allows developers to easily authenticate their users across websites and apps without having to own and manage password files. The end benefit to both the developer and the enterprise is a secure, verifiable way to validate the identity of the person using any app connected to back to the enterprise.

OpenID Connect sits on top of OAuth to verify user identity and user attributes, and the Layer7 implementation is certified in four profiles – Basic, Config, Implicit, and Hybrid.

How Does Layer7 Help for Two or Three-Legged OAuth Use Cases?

The Layer7 API Management can provide both endpoint authorization services and access control for protected services. These two functions can co-exist in a single Gateway or they can be separated out. The benefit of separating them out is around scalability, redundancy and geographic distribution of services. It also allows alignment around business cases, such as physical partitioning of corporate versus public APIs. Most organizations have a large number of APIs to protect, often served from a number of different locations. In these cases, it makes sense to deploy centralized API gateways from Layer7 as authorization servers (often in a cluster for redundancy) and remote clusters of gateways to protect specific API instances.

Both deployment patterns can service OAuth 1.0a and 2.0 versions simultaneously. This pattern also works for both classic two-legged and three-legged scenarios, as well as the OAuth 2.0 grant model, including extension grants such as SAML bearer token. These deployments are illustrated in Figures E and Figure F.

Figure E:

Typical deployment for the classic two-legged scenario or grants such as resource owner credentials and client credentials.

Two-Legged Deployment of CA Technologies

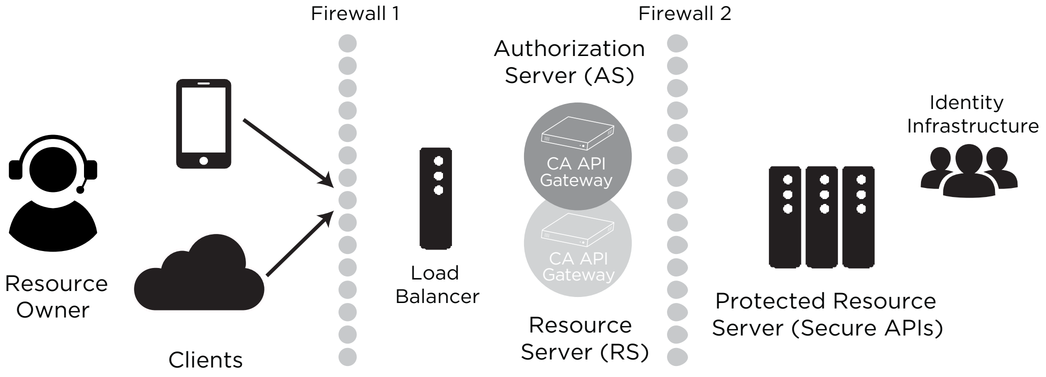
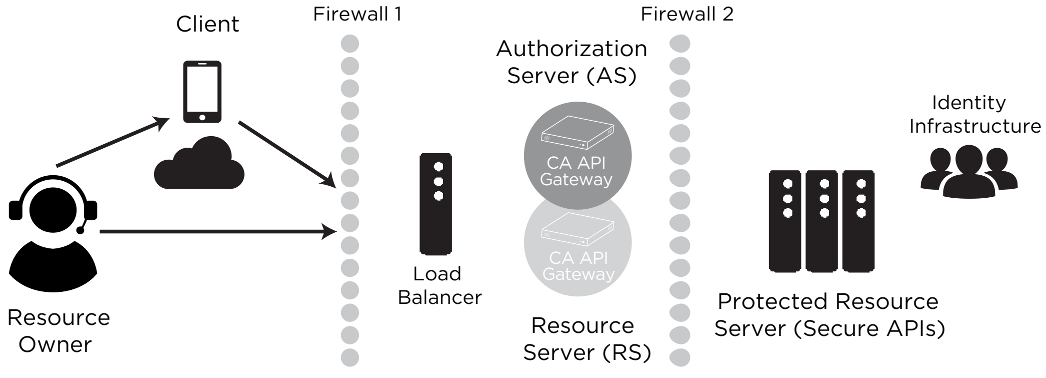


Figure F:

Typical three-legged deployment scenarios and the authorization code grant as well as implicit grant types. Note that Layer7 API Management can simultaneously support all OAuth versions, as well as custom mappings to accommodate interoperability issues.

Three-Legged Deployment of CA Technologies



About Broadcom
Broadcom Inc. (NASDAQ: AVGO) is a global technology leader that designs, develops and supplies a broad range of semiconductor and infrastructure software solutions. Broadcom's category-leading product portfolio serves critical markets including data center, networking, enterprise software, broadband, wireless, storage and industrial. Our solutions include data center networking and storage, enterprise and mainframe software focused on automation, monitoring and security, smartphone components, telecoms and factory automation. For more information, go to www.broadcom.com.

The term "Broadcom" refers to Broadcom Inc. and/or its subsidiaries. For more information, please visit www.broadcom.com. Broadcom reserves the right to make changes without further notice to any products or data herein to improve reliability, function, or design. Information furnished by Broadcom is believed to be accurate and reliable. However, Broadcom does not assume any liability arising out of the application or use of this information, nor the application or use of any product or circuit described herein, neither does it convey any license under its patent rights nor the rights of others.