API Gateways: Where Security Enables Innovation

Version 1.0
Released: September 5, 2013

Securosis, L.L.C. 515 E. Carefree Blvd. Suite #766 Phoenix, AZ 85085 T 602-412-3051 info@securosis.com www.securosis.com

Author's Note

The content in this report was developed independently of any sponsors. It is based on material originally posted on the Securosis blog but has been enhanced, reviewed, and professionally edited.

Special thanks to Chris Pepper for editing and content support.

Contributors

The following individuals contributed significantly to this report through comments on the Securosis blog and follow-on review and conversations (in alphabetic order):

Brian Livingston Patrick McBride

Licensed by Intel Corporation

Intel Expressway Software for API Management, Gateway Security, and Data Protection Compliance

Intel delivers a portfolio of enterprise-class data center software products designed to help expose internal/external app APIs and data across on-prem, cloud, hybrid, and mobile environments. Analyst recognized as the industry leader with both on-prem and SaaS API management portal solutions seamlessly integrated with a Service Gateway for API security, integration and complex traffic control. The Expressway solution includes best of breed Informatics powered data translation, API Management powered by Mashery, mobile access integration with Intel XDK client side tooling, and the only no app impact data tokenization solution for PCI, PII, & cloud data scope reduction. Visit cloudsecurity.intel.com for more.

Copyright

Table of Contents

Introduction 4

Use Case and Deployment 5

Access Provisioning 8

- Provisioning 9

Developer Tools 12

- API Catalog 12
- Deployment Support 13
- Logging, Monitoring, and Reporting 13
- Client Integration 14
- Mobile End to End 14

Key Management 15

- Key Setup and Distribution 15
- Key Lifecycle Management 16

Implementation 18

- Exposing APIs 18
- URL Whitelisting 18

Buyers Guide 20

- Nota Bene: Use Cases First 20

Closing Comments 26

About the Authors 27

About Securosis 28

Introduction

Not long ago, companies designed IT systems to keep people out of their networks. IT Security divided the world between 'insiders' and 'outsiders', granting access to the former while blocking the latter. But the drive to provider better applications to customers, faster than ever before, has caused fundamental changes in how we deliver applications. Companies now *want* to expose their services to a wide audience, but rather than design and build consumer offerings in-house they often provide API access to their services to contractors — and in many cases to the general programming community. For companies like Twitter, Facebook, and YouTube (Google), the trend is to allow third-party developers to extend and integrate these platforms to provide novel new user experiences. It's a win/win: the company gets to leverage innovations from the third-party community, users get better apps, and developers get paid (the average force.com developer makes $392k/year) for their work. It's an *almost* free way to leverage the best ideas in the development world — you just have to accept the risk of random people groping through your services and data.

Stated another way, the traditional DMZ has gone the way of the dodo. Third-party cloud services, mobile devices, and work-from-anywhere employees have destroyed (or at least completely circumvented) the corporate IT 'perimeter' — the 'edge' of the network now contains so many holes that it no longer forms a meaningful boundary. And this trend — fueled by the need to connect customers, partners, and employees to in-house and third-party services — is driving the new model.

Leveraging the community for innovation and free development raises new security problems: how can you control your API while actively making it available outside your company? It is no longer just a web portal — it is your public interface to code and data. Innovation, faster time to market, and better usability all helps land more customers. Not that many years ago companies wrestled with serving up data to consumers outside the firewall — letting their code run on top of proprietary systems is downright scary. To provision developer access, secure the data they access, and to control what they can use and how, you need some form of API management framework. From there, it's easier to get the product in the hands of the end consumers you want using your services.

API gateways have a grand and audacious goal: **enablement**. Getting developers the tools, data, and functionality they need to realize the mobile, social, cloud and other use cases the enterprise wants to deliver. There is tremendous innovation happening today, and the business goal is to *get to market ASAP*. But security is not a nice-to-have — it is a *hard requirement*. After all, the value of mobile, social, and cloud applications is in mixing enterprise functionality inside and outside the enterprise. And riding along is an interesting mix of user personae: customers, employees, and corporate identities, all mingling together in the same pool. API gateways must implement real security policies and protocols to protect enterprise services, brands, and identity. So while the business demands enablement, security teams must reconcile innovation against a reasonable security policy. This is where API gateways come in.

This paper will explore API gateways in detail, straddling the perspectives of buyers who deploy API gateways and developers who use them. Understanding both viewpoints is essential to understanding how to best support development teams and implement a gateway solution.

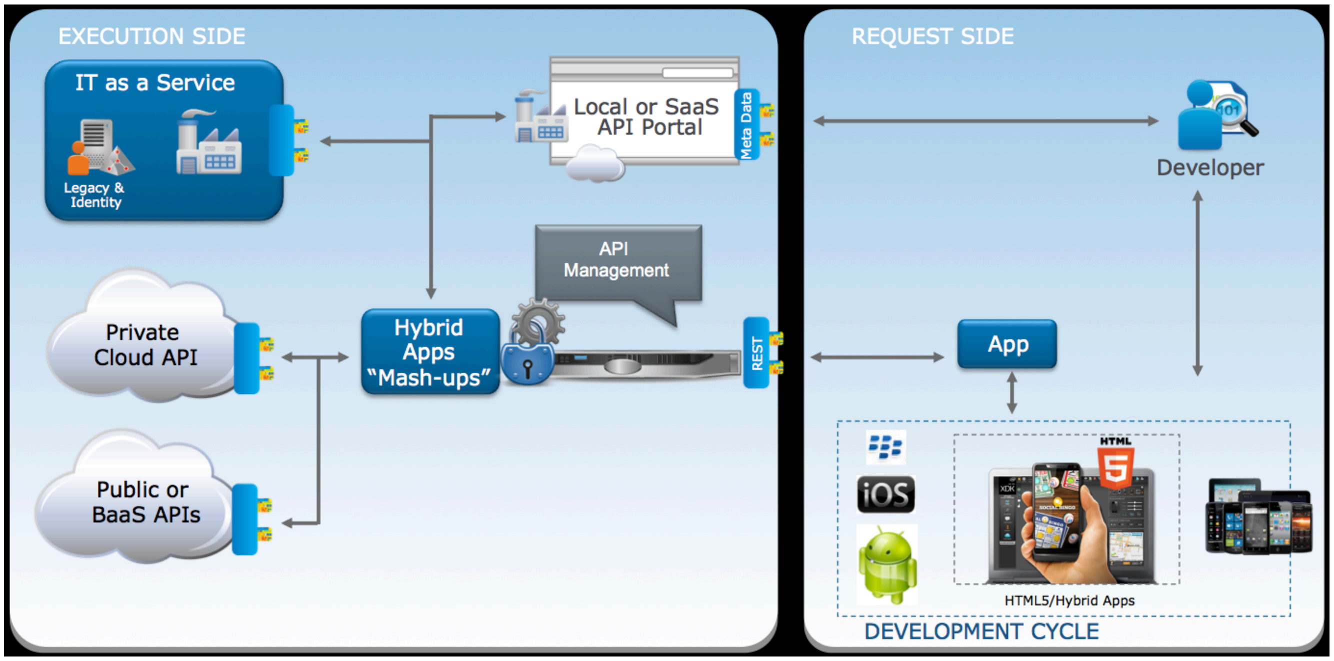
Use Case and Deployment

API gateways enable developers to build cloud, mobile, and social apps on enterprise data, layered over existing IT systems. That said, API gateways are not sexy. They do not generate headlines like cloud, mobile, and big data. But APIs are the convergence point for all these trends and the crux of IT innovation today. We all know cloud services scale almost too well to believe, at prices that seem to good to be true. But APIs are a key part of what makes them so scalable, universal, and cheap. API Mashups bring new ways to collaborate and Mobile apps provide fundamentally new front ends. We are commoditizing our IT systems into open services! Of course open, API-driven, multi-tenant environments bring new risks with their new potentials. As Netflix security architect Jason Chan says, securing your app on Amazon Cloud is like rock climbing — Amazon gives you a rope and belays you, but **you** are on the rock face. You are the one at risk. How do you manage that risk? API gateways play a central role in limiting the cloud's attack surface and centralizing policy enforcement.

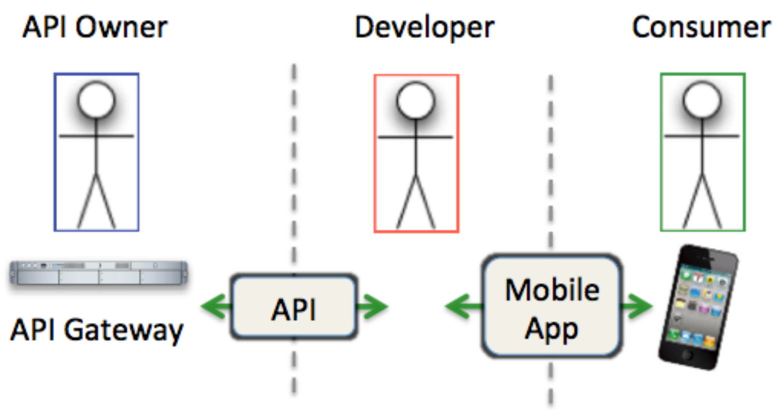
Mobile apps pose similar risks in an entirely different technical environment. There is endless hype about iOS and Android security, but where are the breaches? *On the server side*. Why? Because attackers are pragmatic and that's where the *data* is. Mobile apps have vulnerabilities that attackers can go after one by one, but a breach of server-side APIs exposes the whole enterprise enchilada. Like cloud applications, API gateways need to reduce the enterprise's risk by limiting attack surface. Mobile apps use web services differently than other enterprise applications, communications are mostly asynchronous, and the identity tokens are different — expect to see less SAML or proprietary SSO, and more OAuth and OpenID Connect. API gateways address the challenges raised by these new protocols (often in combination with older protocols) and interactions.

APIs put a simple interface on existing technology, linking new and old applications together into a unified service model. API gateways may be deployed through an appliance, a virtual server, or even as a cloud service that proxies your APIs from a remote location. This way your applications truly become part of a system of services that deliver different flavors and capabilities depending upon the user, the type of device, and the third-party developer's vision for the app. The gateway helps not only provide the glue from the back end to the front, through simple REST based APIs, but the management and metering of the service. In fact, metering — tracking how often users come through the portal — helps gauge how successful a particular implementation is. Often developers are paid by how effective their apps are, and metering traffic is how they are compensated.

The following graphic shows the entire API ecosystem, including the back office systems that execute user requests, the APIs that provide access, and the user environment where requests for service are made:



But keep in mind that while there are a lot of moving pieces, the fundamental relationship is between the owner of the APIs, the developer who extends those APIs to external users, and the consumer. This extension of trust to third-party developers, most unknown to the companies who provide APIs, is a major step forward for large enterprises. But allowing developers to operate outside the formal corporate development processes fosters agility and creativity. And by extension companies quickly deploy onto new platforms, serving customers they would not otherwise not reach.



While cloud, mobile, and other innovations drive radical changes in the data center, one thing remains the same: the speed at which business wants to deploy new services. Fast. Faster! Yesterday, if possible. This makes developer enablement supremely important, and is why we need to weave security into the fabric of development — if it is not integrated at a fundamental level security is all-too-often removed as an *impediment* to shipping. The royal road is to make it easy for developers to understand how to build and deploy an app, grok the interfaces and data, and quickly provision developers and their app users to login — this is how IT shops are organizing teams, projects, and tech stacks.

API gateways curate APIs, provision access to users and developers, and facilitate key management. API Gateway provide services that are not as obvious to developers, but are still extremely useful. A good example here is identity and attribute mapping. A common issue for apps that traverse the gateway is that the external and internal definitions of identity, attributes, token schemes, naming, and authorities do not match. API Gateways provide identity and attribute mapping services that implicitly weave together these identity systems. For security this is the place to focus — to centralize policy enforcement, implement enterprise protocols and standards, and manage attack surface.

When offering access to any back office system, compliance and data governance are top-of-mind concerns for IT security. It often takes a leap of faith for many IT managers to open up access to these critical systems, but the good news is that the gateway provides both a single place to enforce policies but to monitor traffic. All commercial API gateway products provide logging and auditing capabilities, and often incorporate policy enforcement mechanisms over and above API calls. For example, some provide capabilities to screen outbound data, either through tokenization or redaction, dynamically protecting data — based upon defined policies — before it leaves the company. The ability to define data usage policies, over and above data access policies, helps control information and provide necessary controls that auditors want to see.

Access Provisioning

Picture a team of developers sitting around a conference room table with your security team. What are they really asking?

What do we want?

API Access!

When do we want it?

Now!

You want your back-office systems and services available to third party developers so they can find creative new ways to package and present those services to end users, in effect creating new channels to connect to customers. Developers want to leverage your services to make money, and the faster and easier that is the better. But if you are not careful you face a security and compliance nightmare. API gateways are a response to this "open API" movement, designed to bundle your API into an easy-to-use package while addressing security issues. The first step in developer participation is getting them access to the API, and API access is enabled through access tokens.

The trend to build security into the software development lifecycle (SDLC) has been growing over the last decade. The industry focused on what kind of security services to build and how to adopt different SDLC methodologies, but after all these years we have learned that the single biggest part of "security in the SDLC" isn't security and it isn't the SDLC. It's **in.** Getting security **in** the SDLC means deep integration of people, process, and technology, and that is not trivial.

API gateways are this framework. To a developer they function like a traditional development environment: they bundle a number of features under one umbrella, offer basic tools developers need, and make API integration as simple as possible. API providers appreciate this because rich and accessible services encourage developers. Security architects appreciate the ability to centralize policy enforcement at design time and run time. The tougher flip-side is trying to manage developers who don't work for your company, along with giving up control over endpoints and user experience. Additionally, you need to control access and features through tokens and keys. 80% of your API gateway effort will focus on what developers need to leverage your service, but the most difficult 20% will be managing developers' experience and exposing services, which requires focus on ease of use and hiding complexity.

API gateways extend features to developers, so most of our examples are from the development perspective. We work through the path developers take to use your APIs. We start from ground zero, as developers register themselves to use your service, with considerations for how you will provision developer access. Then we move into other areas such as development tools, key management, and other critical areas of API security.

On our journey we straddle two realms: buyers and builders. Buyers are the user you ultimately want using the product, and builders are the developers who create the user experience for your services. It's important to remember that the real goal is to get services to the consumer, but it is the *developer* who uses the gateway to build upon, and so we will focus on the developer experience. We show builders examples of features you need to build into your platform; the easier it is for them to use, the better and faster they deliver. For those of you looking to buy an API gateway, consider this a mirror image of your critical platform criteria and where you need services to get your deployment over the finish line.

Provisioning

As simple as it may seem, provisioning API gateways is a balancing act. On the one hand companies want simple streamlined access for developers to build functionality. On the other hand they want to ensure this all complies with security policies. How can you ensure security while providing developers with full access? What process will ensure the right mix of policy checkpoints without hampering developers? Therein lies the rub.

Let's look at a developer's first step: getting access to the development environment.

Developer access provisioning

Perhaps you have heard that developers can be a tad mercurial? Development is about building and enabling, so security controls which restrict usage or limit functions are seen as an impediment and source of friction. Keeping developers on board with security policy is a challenge, especially when any number of them don't even work for your company. Development tools are typically selected for ease of use, so streamlining access to tools and simplifying access to API functionality is critical.

API gateways proxy communications to applications — acting as traffic cops to direct application requests according to policy. That middle ground is a vital place for security to focus for three reasons:

- It is a boundary between internal and external, making it an ideal place for policy enforcement.
- It is a logical place to monitor inbound and outbound access.
- It is where developers get everything they need to create applications.

What do developers need to get started coding? They need to be vetted to the API, which means they need credentials. These credentials come as tokens and possibly certificates. API gateways should provide what developers need to find and bind to your API to begin coding. First, developers generally need to register with the gateway to initiate the key issuance process and get credentials to your API. This may require a few minutes for a simple automated process, or much longer for requests which require manual review. Once accepted, developers receive credentials — often only to a development and testing instance, with production access to follow.

How this process works, and how simple it is to implement, are important factors for selecting an API gateway. How well can your candidates be tuned to your organizational needs? When building an API gateway be realistic about what developers will tolerate in terms of delay and complexity. Grand processes with many steps tend to stop developers in their tracks.

API documentation is another major factor in simplifying developers' lives. The favorite words of many developers are "for example", followed by a code snippet and a usage explanation. The goal is to get developers up and running quickly so look for code samples, reference implementations, and test clients when you evaluate API gateways. A wide variety of languages are in play, so over time you will likely build your own miniature United Nations — supporting REST, JavaScript, iOS, Android, Java, Python, .NET, and other implementations so developers can learn from frameworks they are comfortable with. Time they spend learning the intricacies of your platform is not spent on building great tools for your users.

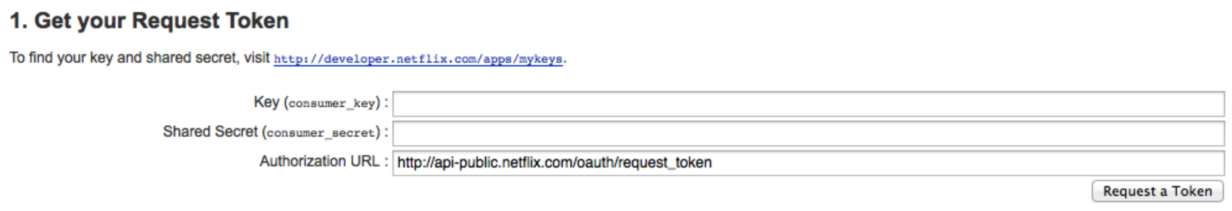
Its worth noting that the flexibility in serving developers also helps when dealing with different lines of business or organizations with different deployment types.

Security Tokens

Access tokens are coin of the realm — they enable just about everything. Developers need to get access tokens, build them into apps, and use them at runtime. As much as security folks wish all developers were fascinated by the deep internals of identity and cryptography, most just don't care. API gateways mean developers do not need to be expert with identity token systems, or with how to route users to the correct services. The gateway's role is to simplify these security steps, and provide users the right view according to their credentials.

Credentials such as OAuth access tokens are critical for API access control. The gateway's job is to provision these tokens to developers so they can build them into applications as needed. Tokens are used by applications at runtime to provide identity for access control, and to facilitate monitoring so no developer can hammer your service too hard.

Just as API gateways show developers how to code clients, they also provide tools to help developers navigate the often byzantine world of identity and access protocols. Netflix's authentication walkthrough is a good example. Short and to the point, their walkthrough leads developers through acquiring and using an access token.


1. Get your Request Token

To find your key and shared secret, visit http://developer.netflix.com/apps/mykeys.

Key (consumer_key) :	
Shared Secret (consumer_secret) :	
Authorization URL :	http://api-public.netflix.com/oauth/request_token

The API gateway should mask many of the complexities of token issuance and management, but developers cannot afford to be wholly ignorant of how tokens provide security. They need at least a basic understanding of identity and access tokens, how they work, and what they do *not* do. Attackers find and abuse gaps between what is expected and what is provided. Take the most common access token: OAuth — called both an authentication token and an authorization token. But in practice it isn't quite either. OAuth provides access tokens which can be used by applications to make access control decisions, such as authorization. Developers should know how to get things working with identity and access tokens, along with their additional responsibilities for implementing a full access control chain. The token is an essential part, but there are many other application and protocol areas to address for safety.

Administration and Policy Provisioning

API gateways are developer focused tools, but they are acquired and managed by companies that wish to provide APIs for public consumption. IT Administrators manage these gateways so that APIs operate according to runtime security policies. Typical policies include version control, monitoring usage, metrics, throttling, and caching.

The API gateway is a registry of application interfaces, which the security team can leverage to create and enforce policies for malicious input such as SQL injection, XSS, CSRF, and other web-app style attacks. The API Gateway can enforce these checks on a pretty granular level, not just message payload but down to the field level.

For application monitoring the API gateway can act as a sensor, collecting inbound and outbound messages. These messages may be tagged with additional context and sent to a SIEM and/or written to an audit log.

Of course there is no silver bullet, and developers are moving quickly to deliver mobile apps and embed new services into web pages. To remain relevant security teams need to be quick too. We need as many tools and as much automation as we can get, attackers evolve and so must defenders. Building around central choke points is immensely useful for gaining visibility and ensuring identity protocols safely control access.

Developer Tools

Now that the developer has access to the API it's time to discuss how the gateway supports your code development and deployment processes. An API gateway must accomplish two primary functions: help developers build, test, and deploy applications; and help companies control use of their API. They serve both as a development environment and an operational security tool.

API Catalog

The API catalog is basically a menu of APIs, services, and support services that provide front-end integration for developers to access back-office applications, external APIs (for mashups), data, and related services, along with all the supporting tools to build and deploy applications. Catalogs typically include APIs, documentation, coding help, build tools, configuration requirements, testing tools, guidance, and sample code for each supported function. They offer other relevant details such as network controls, access controls, integration options, orchestration, brokering, and messaging options — all bundled into a management interface for selecting and configuring desired services. Developer time is expensive so anything that streamlines this process is a win. Security controls such as identity protocols are notoriously difficult to fully grasp and correctly implement. If your security architects want developers to "do it right", this is the place to invest time showing them how.

Traditionally security tools are bolted onto — or in front of — applications, generating howls of displeasure from developers who worried about added complexity and performance impact. With third-party APIs things are different because security is part of their core value. API gateways offer features to enable network, interface, and data security as part of their core feature set. For example it is faster and easier to enable built-in SAML or OAuth identity services than to build them from scratch — or worse, to build a password management system. Even better, the features are available at *design* time, before you assemble the application, so they can be bundled into the development process. Everyone knows not to write your own crypto, don't write your own access control either.

Reference implementations are extremely helpful. For example consider OAuth: if you look at 10 different companies' OAuth implementations you will probably find a dozen different implementations. Don't assume developers will just figure it all out — connect the dots. For a chance at a secure deployment, developers need concrete guidance for security services — especially for things as abstract as identity protocols. Reference implementations show end-to-end examples of the identity protocol in practice. For a developer trying to "do it right", this is like finding diamonds in the backyard. The reference implementation is even more effective if it is backed up by testing tools that can verify developer implementations.

Access management is a principal feature of API gateways. The gateway helps you enforce access controls, building authentication and authorization services into the API set. Gateways typically rely on token-based security services, supporting one or more token services such as SAML and OAuth. All API gateways offer authentication support, and most integrate with other identity sources to support federation. Gateways provide basic role-based authorization support, sometimes with fine-grained authorization to constrain data access by user identity or endpoint device.

Beyond identity protocols, some gateways offer services to defend against attacks — particularly replay attacks and of session hijacking. API gateways provide dynamic filtering of requests, allowing policy-based routing and response to API calls. Developers get tools to parse incoming calls, filter or transform messages, and then route to appropriate services. This facilitates modification of application function, debugging of application functions, and application of different security or compliance controls in response to user requests. Filters also provide a mechanism for sending requests to different locations, workflow modification, or even sending requests to different applications. This flexibility in dynamically adjusting access to services and data is a powerful security capability, particularly for analysis of and protection against suspect clients.

Each API gateway provider offers a range of pre-deployment tools to validate applications prior to deployment. Sandbox testing and runtime simulators validate correct API usage, and can also verify that the application developer properly handles input variables and simulated attacks. Some test harnesses are provided with gateways and others are custom implementations by API service owners. Pre-deployment validation is good a way to ensure all third-party developers meet a minimum security standard and no single user becomes the proverbial weak link. If possible tests should be executed as part of the normal integration process, (*i.e.*, Jenkins) so implementation quality can be continually tested.

Deployment Support

The API catalog provides options for how to build security into your application, but API gateways also offer deployment support. When you push APIs that connect the world to internal systems you need to account for a myriad of different threats at multiple network, protocol, application, and data layers. Denial of service, parser attacks, code injection, replay attacks, HTTP protocol abuse, network sniffing, and denial of service attacks are all things to consider.

API gateways can optionally provide privacy and security for network sessions through SSL. Most also offer network firewall capabilities such as IP whitelisting, blacklisting, and signature-based detection. While network security is a must have for many, it is not really their core security value. The key security features are overall security of the API and message-level filtering. API gateways provide capabilities to detect code injection, cross-site scripting, and various encoding attacks; most also offer off-the-shelf filters for input validation and sanitization.

Logging, Monitoring, and Reporting

As application platforms, API gateways capture activity and generate audit logs. Sitting between developer applications and internal APIs, they are perfectly positioned to capture API usage — for throttling, billing, and metering API access, as well as for security. Log files are essential for security, operations, and compliance, who all rely upon gateway audit trails. Most API gateways provide flexible configuration of which audit events are collected, as well as record format and destination.

Audit capabilities are mostly designed for gateway owners, rather than developers. But the audit trail captures sessions of all users across all third-party code, providing an overview of all activity. Logs generally go into a supplementary SIEM or log management tool for event storage and parsing, using `syslog` or an equivalent format. API gateways provide extensibility to format, tag, and enrich events — with information from the network and API layers, as well as context from the application session — to help SIEM or log management leverage them.

Beyond security audit requirements many deployments have hard 3M requirements for the API Gateway to provide monitoring metering, and metrics. These have obvious overlap with security considerations, but often they can relate directly to line of business functions like payment and allowable usage.

Most API gateways provide rudimentary extensibility for alerting on both usage patterns and suspicious behavior. This type of event monitoring can be used to provide basic security and intrusion detection capabilities out-of-the-box. As described above, message filtering is a more advanced and effective way to detect malicious events, but filtering requires careful development and selection of which capabilities to deploy.

Client Integration

"*In my beginning is my end.*" — T.S. Eliot

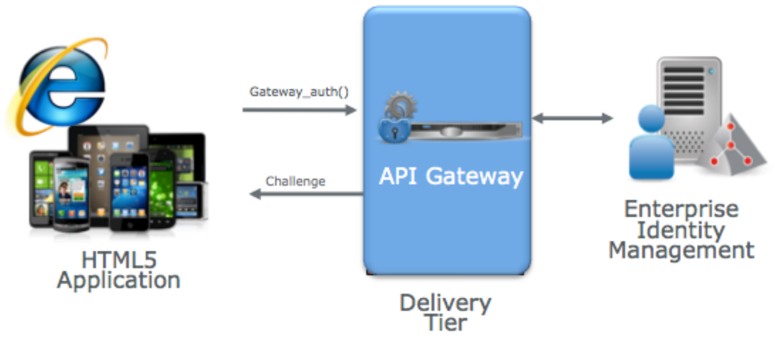
An API is only as useful as the clients it supports. API development in most cases is the art of the possible: supporting all your different kinds of clients. Your API will be called by iOS & Android apps, web service clients, .NET and Windows desktop applications, other APIs, and so on. The fully deployed API gateway is the starting point for client integration. Client integration becomes a game of adapters, wrappers, and domain or technology specific ways to use your API … in iOS, from mashup APIs, and so on.

As always, developers want simple tools to get things working across a broad range of environments. The security challenge is to offer domain-specific guidance suited to the integration context. At a basic level, can you even support the same protocols across all environments? The answer is often 'no'. If not, what does "safe API consumption" look like for each client environment? From there, what kinds of data can be accessed, and where should it reside? How are keys managed and stored? How will you implement SSL and verify the gateway's hostname from each supported client? How can you test this? Unpacking and answering these questions and more shows developers how to use and consume the API and data safely.

Mobile End to End

Mobile clients are an especially important type of client integration. Smartphones, tablets and other devices have unique capabilities and constraints. But of course mobile applications are the combination of the mobile app plus the server. The API Gateway can act as traffic cop, directing clients and enforcing policy for mobile clients.

What's vital to developers is helper classes, SDKs and other libraries that they can package with their Mobile app to communicate with the server, encode and decode on/off the wire, parse data, and implement security policies. These operations are very platform specific (your iOS SSL libs won't work on Android and vice versa). So the most useful solutions for mobile developers are those that are portable across clients (like HTML5 and Javascript) and native implementations (such as iOS and Android SDKs).



Beyond communications and parsing, there are mobile specific considerations like session management, authentication and local storage. The mobile is not an island, its server connection is as critical as an oxygen tank is to a SCUBA diver. The Gateway is the first mile of the mobile app integration, connecting it to enterprise resources, but the last mile must fit cohesively to link the client's needs to the server.

Key Management

For developers one of the most visible API gateway operations is key management. But, dear reader, this is not your father's key management — the kind laden with X.509, PKI, and baroque *foofaraw* security teams had to beg developers to implement. This is 2013 and the keys are *OAuth access keys*! And developers are asking us for the keys too, so what should we do?

Before we answer that question, for those of you who are not programmers, we will describe these *access keys* in some detail. OAuth is a method for authorizing clients (end users and client applications) to use third-party APIs served by the API gateway. It is essentially how developers give access to consumers, without consumers needing to share information such as user name and password. OAuth relies on a trusted identity service to vouch for the client and pass an authorization token to the API, which in turn grants the client access. OAuth enables four parties (a user or consumer, a client application created by a third-party developer, the owner of the APIs, and an identity service provider such as Google or Facebook) to cooperate to deliver services.

As we have discussed, developers are not much keener on the theoretical underpinnings of different identity protocols than the consumers who use their applications. They just want to get users access to the application, so they can move on to more 'meaningful' development tasks… like building the client application itself. This shifts the responsibility of identity and authorization onto security teams, which is a new position for them to be in: managing the process instead of cleaning up afterward. Rather than engaging toward the end of a project to conduct a vulnerability assessment, security teams may select identity protocols to be used, establish identity requirements, and guide developers through the process of building them into their applications. This is an unusual collaboration between developers and security — both in degree and kind. The role of the security team as leader for a portion of the development process sets them up as a true design and development partner.

Key Setup and Distribution

Setting up keys can be handled in several different ways, but the process is typically initiated through self-service features of the gateway — we warned you that it's not your father's PKI. The developer registers their application and client(s). The steps of the OAuth protocol dance vary by implementation, but the core generally includes:

- **Developer account:** A master account for the developer, which can span multiple clients and services.
- **Client ID:** The key that identifies the consumer and grants access.
- **Client secret:** How the consumer authenticates.
- **Client types:** Gateways use these to distinguish between different clients such as iOS and Android devices.
- **Resource:** The URLs, redirects, and other services the client is requesting access to.

Once this bootstrap process is complete — whatever variation your API gateway uses — the client application developer should have everything they need. Once the client has their authorization access token, they are able to call APIs and access data with their token. Each subsequent call to the APIs protected by the API gateway includes an OAuth access token.

The tokens are passed along with every call from the client app to the API so the API can make access control decisions. This is an important part of OAuth's value proposition: the processes of *acquiring* and *using* tokens are kept separate. This means the enterprise security architect must ensure that the policy and governance models for these two independent processes are consistent. Users access only the APIs they are authorized for, and do *not* see other APIs or other users' data. The access rights requested at token issuance must match runtime behavior.

Key Verification Services

Developers may not be interested in identity protocols, but they all care whether their code works. Distributed applications are notoriously difficult to debug, so anything fundamental to operations must be tested. Once access keys are issued and ready for use, the API gateway offer testing tools to ensure there are no surprises at runtime. The API provider must actively help validate client code to protect their API! There are a number of considerations:

- Ensure a production-like system is available for testing. Any networked application must deal with a myriad of issues such as ports, routing, and redirects. A token cannot simply be appended to access and refresh requests — each variant of API usage requires its own test cases.
- Make simple tools available — many APIs include simple cURL scripts to test applications. For example: "`curl https://example.com/API/myservice -H 'Authorization: your OAuth access token'`". Gateways commonly include several scripts to validate client usage of the API.
- Provide documentation and guidance for additional testing and debugging as needed for the client environment.

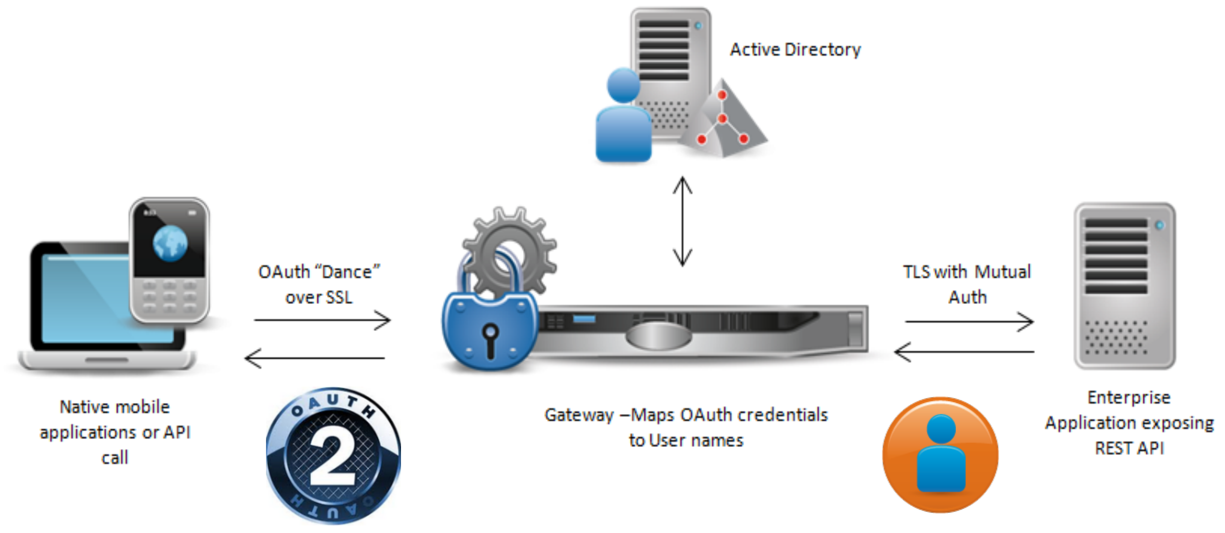
Key Lifecycle Management

OAuth isn't magic security dust, and it doesn't automatically make applications secure. API developers and consumers need to be clear on safe handling of OAuth tokens across their entire lifecycle. Some rules are straightforward, such as to always use TLS/SSL. But most depend on context, such as secure storage for tokens and safe handling of redirects.

Two operations that generally require special attention in security policy are refresh and revocation. OAuth *access* tokens provide short-lived access, but can create long-lived sessions with *refresh* tokens. The refresh token is effectively a protection against an *access* token being replayed. Each consumer may have both different types of tokens. Security policy makers should align policies for both types, taking advantage of the separation between shorter-lived and longer-lived tokens.

In addition to extending sessions, *revocation* requires consideration. Token revocation may sound minor, but anyone who has lost their mobile device can say with authority that it is nice to be able to log into twitter.com and turn off access to your lost mobile phone so its clients no longer offer access to your accounts with cached OAuth keys. Token revocation services provide invalidation for OAuth tokens and help enterprises deal with API consumers which violate policy or otherwise need the plug pulled.

But not every standard enjoys wide adoption or consistent implementation, and attackers break implementations much more often than standards. Many 'standards' are in their infancy, with only limited adoption even among cloud and identity service providers. Further, in a federated relationship both parties must support the same approach to SSO and/or federation for things to work. The following chart shows the relative maturity of several common standards.


Active Directory
OAuth "Dance" over SSL
TLS with Mutual Auth
Native mobile applications or API call
Gateway –Maps OAuth credentials to User names
Enterprise Application exposing REST API
OAUTH 2 OAUTH

Its important to note that OAuth is only one approach, its the most common by far, but its not the only. Your enterprise may use X.509 certificates, SAML, home grown tokens or any number of other implementations. Most enterprises use an external token like OAuth and an internal token like SAML, X.509 or Kerberos (to connect to SOA, ESB and other environments). Similar to identity mapping, the API Gateway can verify the external OAuth token, do the mapping, and issue an internal token. In any scheme, the ability to centralize enforcement while allowing for distributed usage is the goal.

Implementation

APIs go through a software lifecycle just like applications. The purchaser of the API develops, tests, and manages code as before, but when they publish new versions the API gateway comes into play. The gateway is what implements operational polices for APIs — serving as a proxy to enforce security, application throttling, event logging, and routing of API requests.

Exposing APIs and parameters, as the API owner grants access to developers, is a security risk in and of itself. Injection attacks, semantic attacks, and any other way an attacker might manipulate API calls is in play unless you filter requests. Here we focus on implementation of security controls through the API gateway, and how the gateway protects the API.

Exposing APIs

What developers get access to is the first step in securing an API. Some API calls may not be suitable for developers — some features and functions are only appropriate for internal developers or specific partners. In other cases some versions of an API call are out of date, or internal features have been deprecated but must be retained for restricted backward compatibility. The API gateway determines what a developer gets access to, based on their credentials.

The gateway helps developers discover what API calls are available to them — with all the associated documentation, sample scripts, and validation tools. But behind the scenes it also constricts what each developer can see. The gateway exposes new and updated calls to developers, and acts as a proxy layer to reduce the API attack surface. The gateway may expose different API interfaces to developers depending on which credentials they provide and the authorization mapping provided by the API owner. Most gateway providers actually help with the entire production lifecycle of deployment, update, deprecation, and deletion — all based on security and access control settings.

URL Whitelisting

We define *what* an application developer can access when we provision the API; URL whitelisting defines *how* it can be used. It is called a *whitelist* because anything that matches is allowed and non-matching requests are dropped. API gateways filter incoming requests according to rules you set, validating that incoming requests meet *formatting* requirements.

This checking catches and stops some mistakes; the API gateway's security also blocks mistakes by preventing use of unauthorized requests. This may be used to restrict which capabilities are available to different groups of developers, as well as which features are accessible to external requests; the gateway also prevents direct access to back end services.

Incoming API calls run through a series of filters, checking general correctness of request headers and API call format. Calls that are too long, missing parameters, or otherwise clearly fail to meet the specification are filtered out. Most whitelists are implemented as series of filters, which allows the API owner to add checks as needed and tune how API calls are validated. The owner of the API can add or delete filters as desired. Each platform comes with its own pre-defined URL filters, but most customers create and add their own.

Parameter Parsing (injection attacks: XML attacks JSON attacks CSRF)

Attackers target application parameters. Parser bugs are hard to test for, so they are a traditional way to bypass access control and gain unauthorized access to back-end resources. API gateways also provide capabilities to examine user-defined content. “Parameter parsing” is examination of user-supplied content for specified attack signatures, which may identify attacks or API misuse. Content inspection works much like a *blacklist* to identify *known* malicious API usage. Tests typically include regular expression checks of headers and content for SQL injection and cross-site scripting.

Parameters are checked sequentially, one rule at a time. Some platforms provide mechanisms to extend checks programmatically, altering both which checks are performed and how they are parsed depending on the parameters of the API call. For example you might check the contents of an XML stream both for structure and to ensure it does not contain encapsulated binary code. API gateways typically provide packaged policies for content signatures of known malicious parameters, but the API owner determines which policies are deployed.

Buyers Guide

Now it is time to consider the key decision criteria for selecting an API gateway. We offer a set of questions to determine which vendor solutions support your API technically, as well as the features your developers and administrators need. These criteria can be used to check solutions against your design goals and help you walk through the evaluation process.

Nota Bene: Use Cases First!

It is tempting to leap into a solution, but hold up! API development is a major trend and security teams want to help solve API security problems. API gateways have been designed to enable developers to jump in quickly and easily. But there is no generic API security model suitable for all APIs. APIs are a glue layer, so the priorities and drivers for your deployment can only be found by analyzing **your** use cases: what components you are gluing together, from what environment (enterprise, B2B, legacy, etc.), to what environment (mobile, Internet of Things, third-party developers, etc). This analysis provides crucial priority weighting.

Product Architecture	Describe the API gateway's deployment model (software, hardware only, hardware + software, cloud).	
	Describe the scalability model. Does the API gateway scale horizontally or vertically?	
	What out-of-the-box connectors and adapters to other software and cloud services are supported?	
	How are new versions and updates handled?	

	What key features do you believe your product has that distinguishes it from your competitors?	
Access Provisioning & Developer Power Tools	What credentials and tokens does the API gateway support for developers and API consumers?	
	How is access governed? What monitoring, management and metrics features does the API gateway offer?	
	Does the product offer client-side helper SDK libraries (iOS, Android JavaScript, etc.) to simplify API consumer development?	
	Describe a typical "day in the life" of a developer, from registering a new API to production operationalization.	
	Describe out-of-the-box self-service support features for registering new APIs.	
	Describe out-of-the-box self-service support features for acquiring API keys and tokens.	
	Describe out-of-the-box self-service support features for testing APIs.	

	Describe out-of-the-box self-service support features for versioning APIs.	
	Describe how your API catalog helps developers understand the available APIs and how to use them.	
Development	What integration is available for source code and configuration management?	
	For extending the product, what languages and tools are required to develop wrappers, adapters, and extensions?	
	What continuous integration tools (*e.g.,* Jenkins) does your product work with?	
Access Control	How are API consumers authenticated?	
	How are API calls from API consumers authorized?	
	What level of authorization granularity is checked? Please describe where role, group, and attribute level authorization can be enforced?	
	What out-of-the-box features does the API gateway have for access key issuance, distribution, and verification?	

	What out-of-the-box features does the API gateway have for access key lifecycle management?	
	What tools are used to define technical security policy?	
	Describe support for delegated authorization.	
	What identity server functionality is available in the API gateway? *e.g.,* OAuth Authorization Server, OAuth Resource Server, SAML Identity Provider, SAML Relying Party, XACML PEP, XACML PDP, etc.	
	What identity protocol flows are supported and what role does the API gateway play in them?	
Interoperability	What identity protocols and versions are supported (OAuth, SAML)?	
	What directories are supported — Active Directory, LDAP, etc.?	
	What application servers are supported — WebSphere, IIS, Tomcat, SAP, etc.?	

	What service and security gateways are supported — DataPower, Intel, Vordel, Layer7, etc.?	
	Are cloud applications supported? Which ones?	
	Are mobile platforms supported? Which ones?	
Security	Describe support for TLS/SSL.	
	Is client side (“2 way mutually authenticated”) TLS/SSL supported? Please describe.	
	Please describe the API gateway’s support for whitelisting URLs.	
	What out-of-the-box functionality is in place to deal with injection attacks such as SQL injection?	
	How does the product defend against malicious JavaScript?	
	How does the API gateway defend against URL redirection attacks?	
	How does the API gateway defend against replay attacks?	
	What is the product’s internal security model?	
	Is Role-Based Access Control supported? Where?	

	How is access audited?	
Cost model	How is the product licensed?	
	Does cost scale based on number of users, number of servers, or something else?	
	What is the charge for adapters and extensions?	

This checklist offers a starting point for analyzing API gateway options. Review product capabilities to identify the best candidate, keeping in mind that integration is often the most important criterion for successful deployment. It is not as simple as picking the 'best' product — you need to find one that fits *your* architecture, and is amenable to development and operation by your team.

Closing Comments

API security is tremendously complex and context-sensitive — and we have just scratched the surface to give you an overview of current trends and solutions. You can spend your entire career studying the nuances of how all these pieces work together. API security is likely only *part* of your job, so we don't expect you to master everything we have discussed immediately. We hope you find this paper useful and know many of you will have questions on features, technologies, and implementation specifics. If you have any questions or want to discuss your particular situation, feel free to send us a note at info@securosis.com.

About the Authors

Adrian Lane, Analyst and CTO

Adrian Lane is a Senior Security Strategist with 25 years of industry experience. He brings over a decade of C-level executive expertise to the Securosis team. Mr. Lane specializes in database architecture and data security. With extensive experience as a member of the vendor community (including positions at Ingres and Oracle), in addition to time as an IT customer in the CIO role, Adrian brings a business-oriented perspective to security implementations. Prior to joining Securosis, Adrian was CTO at database security firm IPLocks, Vice President of Engineering at Touchpoint, and CTO of the secure payment and digital rights management firm Transactor/Brodia. Adrian also blogs for Dark Reading and is a regular contributor to Information Security Magazine. Mr. Lane is a Computer Science graduate of the University of California at Berkeley with post-graduate work in operating systems at Stanford University.

Gunnar Peterson, Associate Analyst

Gunnar Peterson is a Managing Principal at Arctec Group. He focuses on distributed systems security for large mission critical financial, healthcare, manufacturing, and insurance systems, as well as emerging startups. Mr. Peterson is an internationally recognized software security expert, frequently published, an Associate Editor for IEEE Security & Privacy Journal on Building Security In, a contributor to the SEI and DHS Build Security In portal on software security, a Visiting Scientist at Carnegie Mellon Software Engineering Institute, and an in-demand speaker at security conferences. He maintains a popular information security blog at http://1raindrop.typepad.com.

About Securosis

Securosis, LLC is an independent research and analysis firm dedicated to thought leadership, objectivity, and transparency. Our analysts have all held executive level positions and are dedicated to providing high-value, pragmatic advisory services.

Our services include:

- The Securosis Nexus: The Nexus is an online environment to help you get your job done better and faster. It provides pragmatic research on security topics, telling you exactly what you need to know, backed with industry-leading expert advice to answer your questions. The Nexus was designed to be fast and easy to use, and to get you the information you need as quickly as possible. Access it at <https://nexus.securosis.com/>.
- Primary research publishing: We currently release the vast majority of our research for free through our blog, and archive it in our Research Library. Most of these research documents can be sponsored for distribution on an annual basis. All published materials and presentations meet our strict objectivity requirements and conform with our Totally Transparent Research policy.
- Research products and strategic advisory services for end users: Securosis will be introducing a line of research products and inquiry-based subscription services designed to assist end user organizations in accelerating project and program success. Additional advisory projects are also available, including product selection assistance, technology and architecture strategy, education, security management evaluations, and risk assessment.
- Retainer services for vendors: Although we will accept briefings from anyone, some vendors opt for a tighter, ongoing relationship. We offer a number of flexible retainer packages. Services available as part of a retainer package include market and product analysis and strategy, technology guidance, product evaluation, and merger and acquisition assessment. Even with paid clients, we maintain strict objectivity and confidentiality. More information on our retainer services (PDF) is available.
- External speaking and editorial: Securosis analysts frequently speak at industry events, give online presentations, and write and/or speak for a variety of publications and media.
- Other expert services: Securosis analysts are available for other services as well, including Strategic Advisory Days, Strategy Consulting Engagements, and Investor Services. These tend to be customized to meet a client's particular requirements.

Our clients range from stealth startups to some of the best known technology vendors and end users. Clients include large financial institutions, institutional investors, mid-sized enterprises, and major security vendors.

Additionally, Securosis partners with security testing labs to provide unique product evaluations that combine in-depth technical analysis with high-level product, architecture, and market analysis. For more information about Securosis, visit our website: <https://securosis.com/>.