# Five Simple Strategies for Securing APIs

By Scott Morrison, CA Technologies

technologies

# Contents

# What Are APIs and Are They Worth the Risk?

CA World '14
Invent Tomorrow
CA software drives competitive advantage in the application economy.

## How Do You Make the Enterprise Flexible?

APIs are an emerging technology for integrating applications using web technology. This approach is exploding in popularity because it builds on well-understood techniques and leverages some existing infrastructure.

But it is a mistake to think we can secure APIs using the same methods and technology that we used to secure the conventional, browser-centric web. While it is true that APIs share many of the same threats that plague the web, they are fundamentally different and have an entirely unique risk profile that you need to manage.

This eBooklet provides an overview of these new risks, and offers five simple solutions to counter the common threats. By adopting a secure API architecture from the beginning, organizations can pursue an API strategy more safely and securely — and reap the benefits of agile integration promised by this exciting new technology.

# What is an API?

Since the early days of computing, developers have struggled to make applications communicate. Specialized protocols, such as COM+, CORBA, and even SOAP, emerged over the years, but none were sufficient to meet the need for scale, simplicity, and cross-language functionality.

But the answer was actually in front of us the entire time. The World Wide Web was the first truly scalable, distributed system that tied together disparate platforms with a protocol that was simple to understand and deceptively powerful. The crucial insight was to leverage this success for integrating applications other than the web browser/web server pair for which it was designed.

APIs are the technology behind this approach. APIs allow developers to create an open architecture for sharing functionality and data between applications. **APIs are like windows into an application**—a direct conduit that leads straight into the core functionality and data residing in the heart of the app.

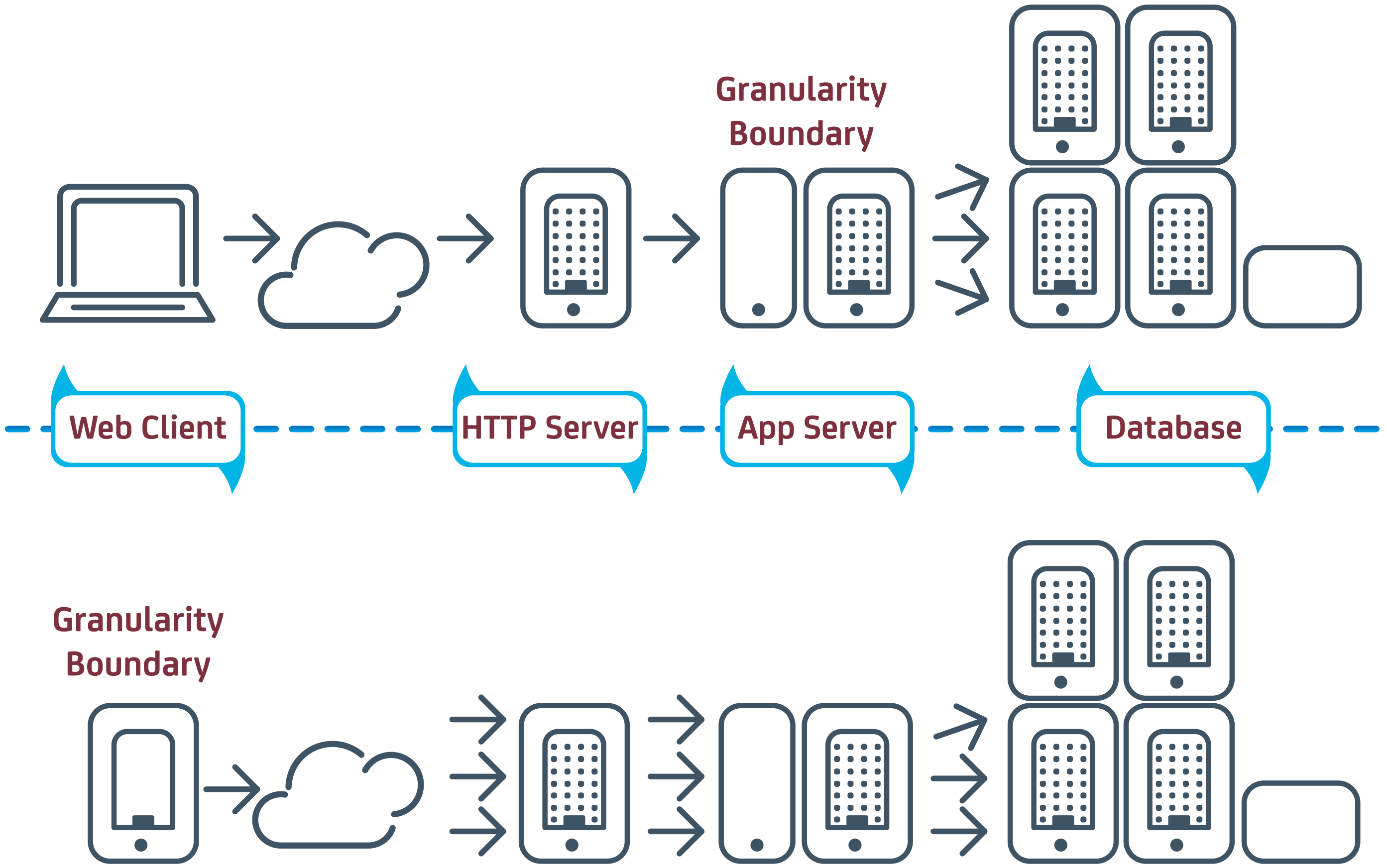
APIs give client-side developers—both legitimate developers and potential system crackers—much more finely grained access into an application than a typical web app. This is because the granularity boundary for calls to back-end tiers moves from relatively secure internal tiers (those that reside safely in a DMZ) all the way out to the client application residing on the Internet.

I know, it sounds like rocket science. But a granularity boundary simply describes how much of the back-end systems a calling application can access. The unfortunate irony is that the same things that make APIs great, also make them a perfect target for hackers.

# How Do APIs Increase an Organization's Risk?

The problem with APIs is that they often provide a roadmap describing the underlying implementation of an application—details that would otherwise be buried under layers of web app functionality. This can give hackers valuable clues that could lead to attack vectors they might otherwise overlook. APIs tend to be extremely clear and self-documenting at their best, providing insight into internal objects and even internal database structure—all valuable intelligence for hackers.

But increased visibility isn't the only risk APIs introduce. Increasing the number of potential calls also increases the attack surface, meaning that a hacker simply has more to exploit. Risk increases with opportunity.



## Old Habits Die Hard

APIs might represent increased risk for the enterprise, but the potential benefits they bring to an organization can overshadow any inherent dangers. The greater threat may be in how we implement APIs.

A well-designed API enables organizations to deliver powerful web tools directly to their employees, clients, and customers. Good API developers understand the threat profile of what they are designing. Unfortunately, many API developers come directly from a web design background, and may bring with them some bad habits. It's important to recognize that despite their common roots and sharing of infrastructure, web design and API design have separate goals and demand different approaches.

There are three main attack vectors that hackers target most frequently with APIs. Understanding these will help you to build safer APIs.

# The Three Attack Vectors to Watch Out For

## Vectors Explained

The most common attack vectors can be broken down into three categories:

### Parameters

Parameter attacks exploit the data sent into an API, including URL, query parameters, HTTP headers, and/or post content.

### Identity

Identity attacks exploit flaws in authentication, authorization, and session tracking. In particular, many of these are the result of migrating bad practices from the web world into API development.

### Man-in-the-Middle

These attacks intercept legitimate transactions and exploit unsigned and/or unencrypted data being sent between the client and the server. They can reveal confidential information (such as personal data), alter a transaction in flight, or even replay legitimate transactions.

## Attack Vector: *Parameters*

Parameter attacks—the most common of which is a SQL injection—attempt to manipulate a system by providing it with inputs that exploit behavior of applications and the infrastructure that supports them (such as databases).

These normally result from developers not carefully checking input into an application. And in contrast to many web apps, APIs often clearly identify a parameter's underlying usage by its name, offering enticing clues to even the casual attacker.

Parameter attacks certainly are not new—the web has been exploited for years using this vector. But they are on the rise in the API world because many developers neglect to sanitize inputs, accustomed as they are to having this applied automatically by many web frameworks. The same risk is associated with APIs, and the same precautions need to be taken to mitigate this threat.

## Attack Vector: *Identity*

We are all familiar with the idea of user identity. But APIs also introduce the concept of application identity—a key that uniquely identifies which application is calling an API. This key, for better or for worse, is commonly called an API key. The API key is replicated across every instance of an application. Its intent is to support basic client management, such as rate limiting, so that an app that goes viral cannot monopolize an API to the detriment of less popular apps.

Unfortunately, API keys are often treated like authoritative credentials, which they are not. API keys are typically hidden inside the code of a calling client application, and despite best efforts to conceal it on the part of developers, they are generally easy to find and exploit. Secure application authentication has always been—and indeed remains—a difficult problem to solve. APIs bring this to light, but unfortunately there are few easy answers to this problem.

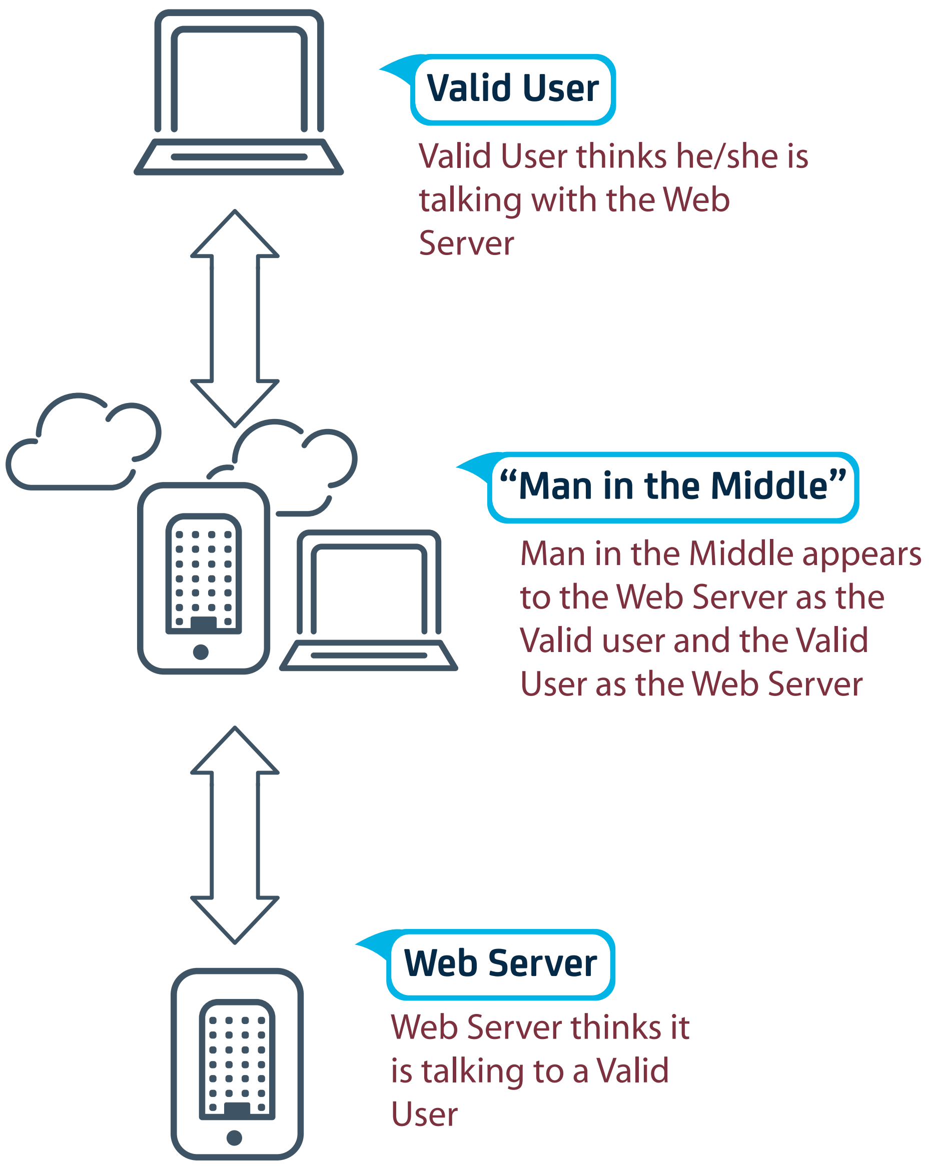
The bottom line is: Never treat API keys as secret.

## Attack Vector: *Man-In-The-Middle*

A man-in-the-middle attack describes a situation in which an attacker sits in between a sender and a receiver of information.

They may do this transparently, or they may explicitly pose as one party or the other, but in both cases they use this position to exploit the exchange of unsigned or unencrypted data.

APIs that are not properly configured using SSL/TLS are highly vulnerable to this form of attack. Unfortunately, the culture in web design is to leave most communications in the clear, largely because of historical challenges in scaling SSL loads. And even when SSL/TLS is applied, often it is misconfigured or vulnerable to downgraded attacks that render it ineffective. In the API world, the stakes are higher and transport protection is essential to secure data, sessions, and access to functionality.


Valid User
Valid User thinks he/she is talking with the Web Server
"Man in the Middle"
Man in the Middle appears to the Web Server as the Valid user and the Valid User as the Web Server
Web Server
Web Server thinks it is talking to a Valid User

## Five Simple Mitigation Strategies

### That Will Allow an Organization to More Securely Publish APIs

Although APIs are susceptible to a broad range of attacks, applying just five simple mitigation strategies will allow an organization to securely publish APIs.

## Strategy 1: Validate Parameters

The first step for any resilient API implementation is to sanitize all incoming data to confirm that it is valid and will not cause harm. The single most effective defense against parameter manipulation and injection attacks is to validate all incoming data against a strict schema—effectively a description of what are considered permissible inputs to the system. Schema validation should be as restrictive as possible, using typing, ranges, sets and even explicit white listing whenever possible. Consider also that the automatically generated schemas produced from many development tools often reduce all parameters to models that are much too broad to be effective at identifying potential threats. Hand-built schemas and white lists are more preferred because developers can constrain inputs based on their understanding of the data model an application expects.

One option for XML-based content types is to use the XML schema language, which is highly effective in creating restricted content models and highly constrained structure. For the increasingly common JSON data types, there are several JSON schema description languages. Although not as rich as XML, JSON is far simpler to compose and understand—offering a transparency which actually makes it simpler to secure.

**Strategy 1:**
Validate Parameters

**Strategy 2:**
Apply Explicit Threat Detetion

**Strategy 3:**
Turn on SSL Everywhere

**Strategy 4:**
Apply Rigorous Authentication and Authorization

**Strategy 5:**
Use Proven Solutions

## Strategy 2: Apply Explicit Threat Detection

Good schema validation can protect against many injection attacks, but consider also explicit scanning for common attack signatures. SQL injection or script injection attacks often betray themselves by following common patterns that are easy to spot by scanning raw input.

Consider also that attacks may take other forms, such as a denial of service (DoS). Leverage networking infrastructure to spot and mitigate network-level DoS assaults, but also check for DoS attacks that exploit parameters. Very large messages, heavily nested data structures, or overly complex data structures can all result in an effective denial-of-service attack that needlessly consumes resources on an affected API server.

Apply virus detection to all potentially risky encoded content. APIs involved in file transfer should decode base64 attachments and submit these to server-grade virus scanning before persisting to a file system where they could be inadvertently activated.

## Strategy 3: Turn on SSL Everywhere

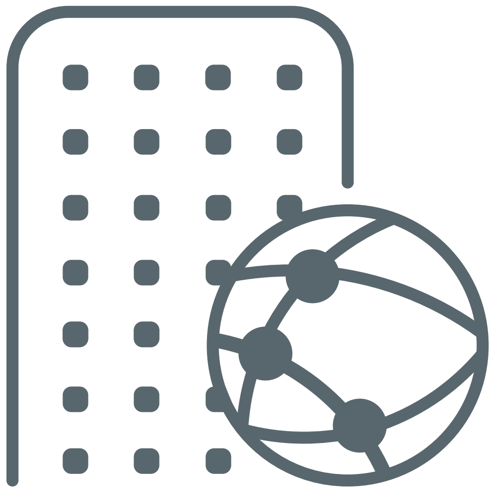
Make SSL/TLS the rule for all APIs. In the 21st century, SSL isn't a luxury; it is a basic requirement. Adding SSL/TLS—and applying this correctly—is an effective defense against the risk of man-in-the-middle attacks.

SSL/TLS provides integrity on all data exchanged between a client and a server, including important access tokens such as those used in OAuth. It optionally provides client-side authentication using certificates, which is important in many environments.

## Strategy 4: Apply Rigorous Authentication and Authorization

User and app identity are concepts that must be implemented and managed separately. Consider authorization based on a broad identity context, including practical factors such as incoming IP address (if known to be fixed or within a particular range), access time windows, device identification (useful for mobile apps), geolocation, etc.

OAuth is quickly becoming the go-to resource for user-centric API authorization, but it still remains a complex, rapidly changing, and difficult technology. Developers should defer to the basic, well-understood OAuth use cases and always use existing libraries rather than trying to build their own.



## Strategy 5: Use Proven Solutions

The first rule of security is: Do not invent your own. There is no reason to create your own API security framework, as there are excellent security solutions that already exist for APIs. The challenge lies in applying them correctly.

### Secure API Architectures

The best way to secure your API from any type of intrusion is to separate out API implementation and API security into distinct tiers. This is a very logical separation of concerns, one that focuses expertise on the right problem at the right time.

This approach frees an API developer to focus completely on the application domain, ensuring that each API is well-designed and promotes integration between different apps. Security then falls into the domain of the expert, who can focus solely on identity, threats, and data security.

# Conclusion

APIs represent a great opportunity for the enterprise to integrate applications quickly and easily. But APIs can be a double-edged sword: promising agility, while at the same time increasing risk. But if an organization can address API security as an architectural challenge long before any development takes place, it can reap the rewards of this technological breakthrough safely and securely.

## Learn more about the inherent risks and benefits of APIs with these resources:

**www.ca.com/techinsights/security**
Leverage industry leading white papers, research and industry reports to gain insight into today's top security trends.

**www.ca.com/api**
Learn how to securely connect your enterprise to mobile apps, cloud platforms and developer networks through APIs.