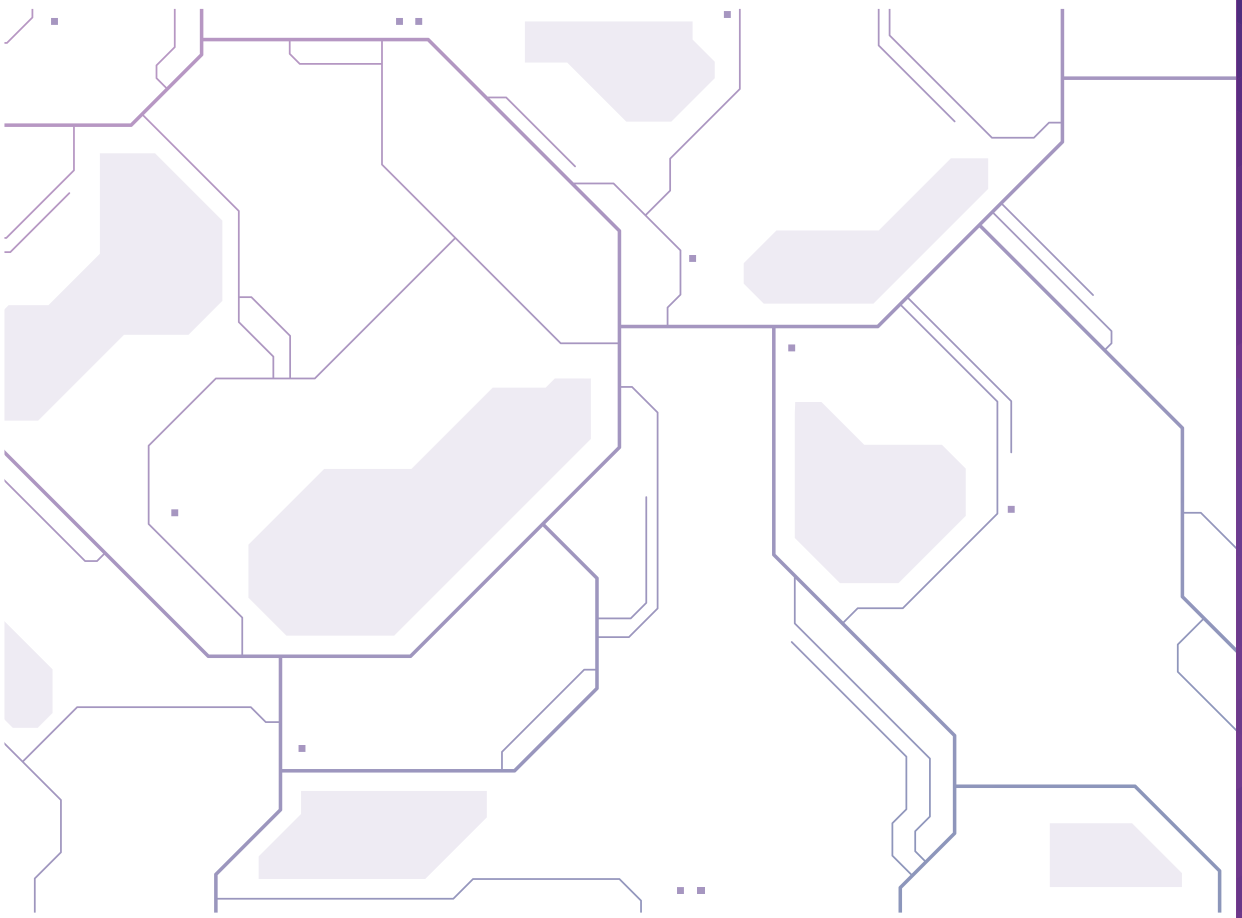
42crunch

# DAST vs API Contract Testing

What **Security Teams** Need to Know



## Executive Summary

As APIs become the primary interface for modern applications, they have also emerged as a top target for attackers. The rise of AI-powered automation accelerates this trend, as the same tools that help developers build APIs are used by attackers to rapidly discover and exploit vulnerabilities.

Dynamic Application Security Testing (DAST) is a long-established method for identifying vulnerabilities in web applications. It works by probing applications with a wide range of attack patterns and monitoring responses to detect signs of weaknesses like SQL injection, XSS, and misconfigurations. DAST tools operate from the perspective of an external attacker, making no assumptions about the underlying code or structure of the application.

Originally designed for traditional web application security, DAST tools are increasingly being repurposed for API security. However, APIs present a fundamentally different challenge.

This paper explains why legacy approaches like DAST often fall short for API security testing, and how API contract testing offers a more accurate, scalable, and effective approach. Readers will gain a clear understanding of how contract-based API security testing works, why it's better suited to today's API threat landscape, and how it enables more reliable security outcomes with less noise and effort.

## Table of Contents

## How API Contract Testing Works

API contract testing finds vulnerabilities more efficiently and accurately than DAST by focusing on whether an API enforces essential security controls. This matters because many API vulnerabilities stem from **missing or weak enforcement of security controls**: things like accepting malicious input, failing to validate IDs or parameters, or not restricting user actions based on roles or permissions. These API implementation flaws lead directly to the most common causes of serious breaches and data leaks, such as:

- **Unauthorized access** to sensitive data (e.g., broken object-level authorization)
- **Privilege escalation** via business logic flaws
- **Injection attacks** (e.g., SQL/NoSQL)

Contract testing validates whether the API has implemented security controls to reject anything it shouldn't accept, and to block anything it shouldn't expose. It uses the contract as a **precise, authoritative definition** of how the API is supposed to behave, providing a reliable and accurate basis to identify incorrect or anomalous API behavior during testing.

In contrast, traditional DAST tools don't know how each API is supposed to behave. This approach relies on throwing thousands of generic attack payloads at your API, most of which aren't relevant to the way your API actually works. And to determine if the API shows signs of vulnerabilities, DAST uses statistical heuristics, comparing observed responses to baseline characteristics, in an attempt to guess whether something is wrong. This approach is inherently unreliable, especially with APIs, and frequently produces false positives or misses critical issues entirely, ultimately generating more noise than actionable insights.

## Comparing DAST and API Contract Testing

| Feature | DAST | API Contract Testing |
|---|---|---|
| Perspective | Attacker's view | Developer's and security engineer's view |
| Methodology | Generic vulnerability tests and statistical heuristics | Policy-driven testing and validation against explicit API contract rules |
| Precision | Prone to false positives/ negatives | High accuracy with minimal noise |
| Dependency | Volume and relevance of tests | Completeness and correctness of the API contract |
| Governance Benefit | None | Contract-driven design, implementation, testing, and documentation |

## Generic vs API-Aware Security Testing

Let's look at some examples to contrast these different approaches to API security testing.

***Example 1: NoSQL Injection***

Good Payload

```
{
  "accountId": "6860d5ebc3dde949d8d1e547,
  "iban": "UK8979879DDDD312349012D1",
  "amount": 2500,
  "currency": "EUR",
  "description": "personal payment"
}
```

Attack Payload

```
{
  "accountId": {
    "$ne": "6860d5ebc3dde949d8d1e547"
  },
  "iban": "UK8979879DDDD312349012D1",
  "amount": 2500,
  "currency": "EUR",
  "description": "personal payment"
}
```

**DAST Approach**: A DAST tool attempts hundreds of NoSQL injection patterns to try to bypass access control or exploit query logic. But with modern APIs exposing rich and complex data structures, and teams using a wide variety of database technologies, **the DAST tool may never stumble upon the exact payload that successfully exploits the vulnerability**. This creates false results, and a false sense of security.

**API Contract Testing Approach**: Rather than trying to exhaust all possible attack combinations, contract testing defines and tests for what should be accepted. A strong contract can specify that accountId is a required string matching a UUID format. Contract tests can then verify that the API rejects any accountId that doesn't match the required format, thereby protecting the API from the NoSQL injection attack.

This makes API contract testing significantly more powerful and efficient. By validating conformance to the contract, the API becomes structurally resistant to a wide range of injection attacks, regardless of whether it uses a SQL or NoSQL backend. It focuses testing not on spotting known threats, but on enforcing proper input structure and behavior, leading to more consistent, scalable, and effective API security testing, not to mention better APIs.

### *Example 2: Business Logic Exploit*

Good Payload

```
{
  "Organization": {
    "TypeCode": 3,
    "Name": "name"
  }
}
```

Attack Payload

```
{
  "Organization": {
    "TypeCode": 0,
    "Name": "name"
  }
}
```

In this scenario, taken from a real-world API incident, a user-supplied input value of TypeCode: 0 results in unintended elevated privileges (e.g., administrative access). This is a **business logic flaw**, not something detectable by generic security scanners.

**DAST tools cannot know the rules** of valid input values for a given service. Unless the tool has specific, custom-built logic for that exact API, it will miss this vulnerability entirely.

**API Contract Testing**, however, can define and validate strict rules for the underlying service to indicate TypeCode must be an integer between 1 and 4. Any deviation in the API's behavior, including for an attack payload like the one above, is flagged as a violation of the contract; a sign that the API's security controls are missing or incomplete.

This example highlights the need for API security tests to be aware of the API context, something only designers or developers with their API contracts can accurately define.

## Test Relevance Matters over Test Volume

In traditional application security, broad test coverage is often achieved by running large suites of generic attack payloads. But in API security testing, **relevance is more important than volume**, and in fact irrelevant tests often do more harm than good.

Take for example one DAST tool we reviewed that ran over 20,000 SQL injection payloads **against an API using a NoSQL database**. Most of those tests aren't just ineffective, they generate noise, consume time and resources, and produce misleading alerts that distract teams from real issues.

The issue here is about context. DAST tools operate without awareness of an API's structure, data model, security requirements, or intent. Instead, generic security rules come pre-built with the tool, inherently lacking context or relevance to the APIs under test.

**API contract testing flips that model**. Tests are derived directly from the API's own design, captured in its API contract, ensuring each test:

- Validates actual input types, formats, and constraints unique to each API
- Checks for required fields and authorization controls
- Verifies output structure and error handling tailored to each API

Every test is relevant, targeted, and aligned with how the API is supposed to function. This leads to higher precision, lower noise, and faster identification of real security gaps, all without wasting effort on irrelevant test results.

## Addressing a Common Criticism: Contract Quality

One criticism of API contract testing is that it relies heavily on the **completeness and correctness** of the API contract. If the contract is not maintained alongside API development, or is missing fields, constraints, or authorization rules, the testing will be blind to those gaps.

This is a valid concern and is often overlooked by tool vendors claiming to provide API contract testing. Without guarantees around contract quality, their results cannot be trusted.

**42Crunch uniquely solves this as a systematic governance process**, providing the tooling to:

- Automatically analyze API contract quality
- Flag incomplete or missing fields (e.g., missing types, formats, authorization scopes)
- Enforce consistent contract authoring across teams
- Guide developers to build complete, enforceable API contracts

By implementing this automated governance approach, practitioners not only establish a solid base for effective API security testing but at the same time also resolve key issues around API

discovery and incomplete inventories, ensure high-quality documentation for API consumers facilitating smoother integrations, and reduce friction between front-end and back-end teams.

Ensuring the completeness and correctness of API contracts is a vital quality engineering practice, one that delivers measurable benefits by producing APIs that are more secure, reliable, and easier to maintain.

## Misconceptions Around DAST and OpenAPI Support

Some DAST vendors market their tools as "supporting API contract testing" by offering add-on features to import Swagger or OpenAPI files. However, in most cases, this "support" is superficial. The imported specification is typically used for two purposes only:

- **Endpoint discovery** – crawling the API definition to identify URLs to probe
- **Generating sample requests** – extracting example payloads or parameters from the spec to seed baseline tests

While these features may help a DAST tool send more targeted requests, they are not the same as validating an API against its contract. In reality, the core testing approach remains unchanged: DAST tools still rely on trial-and-error probing, sending large volumes of generic payloads and using statistical heuristics to guess when a vulnerability might exist.

The result is a clear distinction: importing an OpenAPI file into a DAST tool might make the scanning process slightly more informed, but it is not API contract testing and it will not deliver the accuracy, coverage, or governance benefits that dedicated API contract testing provides.

## Summary: Why API Contract Testing Excels

- **High Precision, Low Noise**: Rather than flooding APIs with generic attack payloads, contract testing verifies an API's security controls as defined in a precise API contract.
- **Relevant Tests**: Tests derived automatically from an API contract are tailored to the fine-grained data and security requirements of each individual API.
- **Developer-First**: Speaks the language of the API development teams: endpoints, schemas, data validation, and access control, not esoteric vulnerability names.
- **Fewer Tests, More Coverage**: A tight contract automatically deflects a wide class of attacks, so developers and testers can do a lot more with a lot less testing.
- **Actionable Feedback**: Results clearly indicate where the implementation diverges from intended security design, helping teams catch and fix API drift early on.
- **Governance Support**: a contract testing approach promotes better documentation, standardization, visibility and collaboration across internal and external API teams.

## Introducing 42Crunch API Scan for Contract Security Testing

42Crunch API Scan is a leading API security testing tool designed specifically for automated API contract testing. Built to address the unique challenges of secure API development, API Scan offers seamless integration with popular IDEs and CI/CD pipelines, enabling developers and security teams to embed security checks directly into their workflows.

One of API Scan's standout features is its intuitive user interface, which allows one-click generation of security tests directly from inside a Swagger or OpenAPI file. This developer-friendly experience accelerates the creation and execution of contract-based security tests without requiring extensive manual setup.

Beyond the developer environment, API Scan excels in automation with advanced CI/CD integration capabilities. It can automatically trigger security testing on pull requests, ensuring that any updates to the API implementation or its contract are immediately validated for security compliance before merging. This proactive approach helps teams catch vulnerabilities early in the software delivery lifecycle, reducing risk and improving overall API security posture.

As the industry's purpose-built tool for API contract security testing, API Scan empowers organizations to maintain robust API security through continuous, automated, and developer-friendly workflows.

### Enhanced API Governance with API Scan and API Audit

To deliver a comprehensive API security and quality solution, 42Crunch offers **API Scan** alongside **API Audit**, a powerful static analysis tool focused on evaluating the quality and security posture of API contracts themselves. While API Scan performs dynamic contract testing against live API implementations, API Audit scrutinizes the OpenAPI specifications to identify potential design flaws, security misconfigurations, and best practice violations before any code is deployed.

When used together, API Scan and API Audit provide a robust, end-to-end governance framework that ensures APIs are **designed, implemented, and tested** according to the highest security standards. This comprehensive approach enables organizations to enforce consistent API design quality and security policies early in the development lifecycle, while continuously validating actual API behavior against these standards in real time.

By integrating both tools into development and CI/CD workflows, teams can confidently deliver secure, well-governed APIs that meet compliance requirements and withstand evolving security threats, all while streamlining collaboration between security, development, and operations teams.

## About 42Crunch

42Crunch enables a standardized approach to API security that automates the enforcement of security compliance across distributed development and security ecosystems. Our API security testing and runtime threat protection services are used by Fortune 500 enterprises, mid-sized firms and more than 1.8 million developers worldwide.

The 42Crunch API security platform empowers developers to build security from the IDE into the API pipeline and gives application security teams full governance from the CI/CD across the entire API lifecycle. This seamless DevSecOps approach to API security reduces governance costs and accelerates the delivery of secure APIs.

### Contact Details

Email: **sales@42crunch.com**
Website: **42crunch.com**