API Testing:
Challenges and Best Practices
Modern composite applications are aggregating and consuming private, partner, and public APIs at a staggering pace in order to achieve business goals. ProgrammableWeb reports that there are over 10,000 APIs published today—which is well over twice as many than there were two years ago. Beyond these publicly-exposed APIs, the number of private APIs is estimated to be in the millions.

As the risks associated with application failure have broader business impacts, the integrity of the APIs you produce and consume is now more important than ever. An API that fails to deliver the expected level of security, reliability, and performance can thus have tremendous business impacts—both to the organization producing it and to those consuming it.

If you are integrating exposed APIs into your business critical transactions, you are essentially assuming the risks associated with that API's integrity (or lack thereof). As the number of external APIs integrated into a business process increases, so do the potential points of failure. The business impact of any application failure is the same, regardless of whether the fault lies within the components you developed or the APIs you are consuming. Finger pointing does little to foster customer satisfaction and brand loyalty.

If you are exposing an API, the assumption is that it will work as described. Once the organizations consuming that API integrate this exposed functionality into their own applications, API failure jeopardizes the transactions that now depend on this functionality. If your API is popular, you can guarantee that a glitch will make the headlines. The more secure, reliable, and dependable your API, the better the chance of consumption and the greater the potential for business expansion. If you're providing a questionable interface and there are viable alternatives to your API, you're likely to lose business since switching costs associated with API integration are so low.

4 Integrity Challenges with the API Economy

In today's API economy, your ability to protect your brand hinges upon your awareness of (and response to) the following challenges...

1. Broader Attack Surface Area

Simply exposing an API through an internal infrastructure undeniably increases an application’s attack surface area from a security perspective. It could be vulnerable to API-level attacks (injections, payload-based attacks, etc.) as well as exploits that take advantage of ineffective authentication, encryption, and access control. The challenge of broader attack surface area is compounded if the API is hosted by a public cloud service. In traditional computing, the producer is aware of (and has full control over) the parameters of the network security. With cloud services, this level of control is significantly diminished. If you’re now leveraging a third-party set of services, the onus is upon you to ensure that your APIs will provide the level of security that your organization expects.

2. Elevated Potential for Unexpected Misuse

Considering the range and number of people who will have access to published APIs, it’s virtually inevitable that they will be exercised in a number of unexpected ways: both by people innocently using them in ways the producer never anticipated and by attackers maliciously trying to exploit them. In the early days of SOA—when services were exposed internally through controlled networks—you could be fairly certain that your services would be used by colleagues or partners who were familiar with their intended use cases. Now, when an API is exposed to the public, the producer surrenders all control and certainty over how those APIs are consumed.

3. Exceptionally Unpredictable Demand

APIs typically need to meet established performance SLAs; however, validating performance vs. SLAs is complicated because it’s so difficult to predict how and when the API might be accessed. Thus, it’s important to validate performance SLAs against a broad range of performance scenarios, including the sudden surges that could occur if an API garners unexpected attention. Moreover, if you’re testing an API that interacts with additional layers of services, the potential for variable or unacceptable performance increases exponentially. This makes it both more critical—and more challenging—to execute a robust set of performance testing scenarios and pinpoint whether the system under test satisfies expectations.

4. API Consumers Need Test Environments

To promote widespread adoption of APIs, it’s often desirable to provide API consumers test environments (a.k.a. sandboxes) that enable them to develop and test against exposed services with zero impact on the production system. Such test environments are also critical when API producers want to jumpstart adoption by allowing integration to begin before their API is actually implemented or before new features are fully completed. Another key driver for providing a test environment is to give API consumers easy access to the broad range of behavior, data, and performance profiles they might want to leverage for testing their integration with that API.

5 API Testing “Must Haves”

Ensuring that APIs are delivering the necessary level of security, reliability, and performance that’s vital to success in today’s API ecosystem inevitably involves developing, continuously executing, and religiously maintaining a broad array of complex tests. Following are several key API testing “must haves” that will help you achieve those goals in light of the above challenges.

1. Intelligent Test Creation and Automated Validation

With APIs, testing a broad range of conditions and corner cases is critical, so automation comes to the forefront. The creation and execution of simple automated tests with limited or manual validation might have sufficed for internal given web services that were used internally (e.g., via SOA), but more sophisticated and extensive automation is required to be confident that APIs are robust enough to meet business expectations. You need a level of automation that gives you a comprehensive set of functional test cases that can be repeated in a systematic manner.
Recommended capabilities for this goal include an intuitive interface for automating complex scenarios across the messaging layer, ESBs, databases, and mainframes:

- Defining automated test scenarios across the broad range of protocols and message types used in APIs: REST, WADL, JSON, MQ, JMS, EDI, fixed-length messages, etc.
- Automating rich multilayer validation across multiple endpoints involved in end-to-end test scenarios.
- Parameterizing test messages, validations, and configurations from data sources, values extracted from test scenarios, or variables.
- Defining sophisticated test flow logic without requiring scripting.
- Visualizing how messages and events flow through distributed architectures as tests execute.

These are all capabilities that should—or at least could—have been applied to web service testing for SOA. In fact, most of these capabilities were invented, tested, and refined in the context of SOA testing. APIs—with their extreme exposure and myriad opportunities for misuse—brings us to the tipping point that makes these automated testing and validation capabilities a "must have" for organizations serious about delivering APIs that satisfy user needs and expectations.

2. Change Management for Test Assets and Environments

Continuously evolving APIs helps organizations stay a step ahead of the competition while responding to business demands. Yet, this frequent change presents significant quality risks if the automated test suite fails to keep pace with the evolving API.

A system for fast, easy, and accurate updating of test assets is critical for keeping test assets in sync with the changing API. If you can automatically assess the impact of changes to existing tests and then quickly update existing tests (or create new ones) in response to the identified change impacts, you can vastly reduce the amount of time required to ensure that your tests don’t fail due to expected changes…or overlook critical new functionality.

3. Service Virtualization for Simulated Test Environments

Service Virtualization technology creates simulated test environments that provide anytime, anywhere access to the behavior of dependent resources that are unavailable, difficult to access, or difficult to configure for development or testing. "Dependent resources" might include mainframes, mobile app front-ends, databases, web services, third-party applications, or other systems that are out of your team’s direct control. Service virtualization can be used in conjunction with hardware/OS virtualization to access the environments you need to test earlier, faster, or more completely.

In the context of API testing, service virtualization can be applied in two key ways:

- To simulate access to the dependent resource behavior (e.g., from a mobile app, database, legacy system, or third-party service) that you need in order to thoroughly validate your API.
• To simulate the behavior of your APIs, creating a test environment that API consumers can develop and test against without impacting your production environment—or to enable development and testing to begin before APIs are completed.

4. Extensive Performance Testing—Ideally, with Service Virtualization

Due to the highly-exposed nature of APIs, there’s a high potential for unpredictable and often volatile traffic volumes. To determine whether your API will satisfy SLAs in the event of the erratic or surging demand that APIs commonly face, it’s essential to ramp up the scope of performance testing. You can use service virtualization (covered above) to create simulated test environments that help you test against different performance scenarios that would otherwise be difficult to create in the test environment.

For instance, you can easily set performance conditions (e.g., timing, latency, delay) to emulate peak, expected, and slow performance—perhaps to help you plan for cloud bursts or determine how the API might respond when someone is accessing it from China. You can also configure various error and failure conditions that are difficult to reproduce or replicate with real systems—for instance, if your APIs rely on Amazon Web Services, you can easily simulate a scenario where AWS is down. This ability to rapidly configure a broad range of conditions in dependent systems is essential for determining if your APIs provide reasonable responses—or at least fail gracefully—under exceptional conditions.

One final way that adopting service virtualization helps performance testing: you can “virtualize” any connections to third-party systems, reliably eliminating the risk that your stress tests might impact services you aren’t permitted (or budgeted) to barrage with test messages.

5. Extensive Security Testing—Ideally, with Service Virtualization

Considering APIs’ increased attack surface area, a multi-faceted security testing strategy is essential for ensuring that development has built the appropriate level of security into your application. This includes:

• Executing complex authentication, encryption, and access control test scenarios.

• Generating a broad range of penetration attack scenarios involving parameter fuzzing, injections, large payloads, etc.

• Running penetration attack scenarios against your existing functional test scenarios.

• Monitoring the back-end during test execution in order to determine whether security is actually compromised.

In addition, if you’re adopting service virtualization (covered above) you can leverage it to take your security testing to the next level:

• It provides rapid ways to emulate attack scenarios as well as emulate different security behaviors of dependencies. This lets you derive more value from your existing functional test scenarios (since you can run them vs. different security scenarios that would otherwise be difficult to configure and unfeasible to test against).

• It enables extensive security testing to be performed without a security expert. Existing test scenarios can be easily executed against a broad set of preconfigured security scenarios.

• It helps you isolate and zero in on your APIs response to various attack scenarios and different security behaviors of dependencies.
About Parasoft

For 25 years, Parasoft has researched and developed software solutions that help organizations define and deliver defect-free software efficiently. By integrating **Development Testing**, **cloud/API testing**, and **service virtualization**, we reduce the time, effort, and cost of delivering secure, reliable, and compliant software. Parasoft's enterprise and embedded development solutions are the industry's most comprehensive—including static analysis, unit testing, requirements traceability, functional & load testing, dev/test environment management, and more. The majority of Fortune 500 companies rely on Parasoft in order to produce top-quality software consistently and efficiently. For more information, visit the [Parasoft web site](http://www.parasoft.com) and [ALM Best Practices blog](http://www.parasoft.com/alm/best-practices).

Contacting Parasoft

<table>
<thead>
<tr>
<th>Region</th>
<th>Phone</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>USA</td>
<td>(888) 305-0041</td>
<td><a href="mailto:info@parasoft.com">info@parasoft.com</a></td>
</tr>
<tr>
<td>NORDICS</td>
<td>+31-70-3922000</td>
<td><a href="mailto:info@parasoft.nl">info@parasoft.nl</a></td>
</tr>
<tr>
<td>GERMANY</td>
<td>+49 731 880309-0</td>
<td><a href="mailto:info-de@parasoft.com">info-de@parasoft.com</a></td>
</tr>
<tr>
<td>POLAND</td>
<td>+48 12 290 91 01</td>
<td><a href="mailto:info-pl@parasoft.com">info-pl@parasoft.com</a></td>
</tr>
<tr>
<td>UK</td>
<td>+44 (0)208 263 6005</td>
<td><a href="mailto:sales@parasoft-uk.com">sales@parasoft-uk.com</a></td>
</tr>
<tr>
<td>FRANCE</td>
<td>(33 1) 64 89 26 00</td>
<td><a href="mailto:sales@parasoft-fr.com">sales@parasoft-fr.com</a></td>
</tr>
<tr>
<td>ITALY</td>
<td>(+39) 06 96 03 86 74</td>
<td><a href="mailto:c.soulat@parasoft-fr.com">c.soulat@parasoft-fr.com</a></td>
</tr>
<tr>
<td>OTHER</td>
<td>See <a href="http://www.parasoft.com/contacts">http://www.parasoft.com/contacts</a></td>
<td></td>
</tr>
</tbody>
</table>

Author Information

This paper was written by:

- Wayne Ariola (wayne.ariola@parasoft.com), VP of Strategy at Parasoft
- Cynthia Dunlop (cynthia.dunlop@parasoft.com), Lead Technical Writer at Parasoft