API Testing Methodology — Where Synack API Penetration Testing Fits In

This document reviews different types of API testing, clarifies what Synack provides in API pentesting and details researcher methodologies. Please note that although all of the testing types described in this document have a valuable place in your API development, not all are addressed by Synack.

What Synack provides: Security Testing and Penetration Testing

Synack solves one aspect of API security by offering API pentesting, performed by researchers of the Synack Red Team (SRT).

Synack API pentesting provides a true adversarial perspective, as researchers attempt to exploit the API in the way a real external adversary would. Their human intelligence and creativity go beyond that of automated testing solutions, using the documentation to enable the most effective test possible. SRT researchers will attempt to exploit common and critical vulnerabilities, including (but not limited to) a subset of the OWASP API Top 10.

Understanding the OWASP API Top 10 vulnerabilities can paint a clear picture of Synack researcher methodology. Here, we enumerate the Top 10, articulating the definition of the flaw and clarifying how it fits into a Synack test. Note that only 7 of the 10 are applicable to Synack API Pentesting.

Researchers are not limited to the OWASP Top 10; it is simply an effective list for providing general guidance and understanding of testing methodology.

OWASP Top 10 API flaws

Broken Object Level Authorization ●	Attackers manipulate object identifiers within a request to gain unauthorized access to sensitive data. This should sound familiar to security practitioners; it's referred to as an Insecure Direct Object Reference (IDOR) in a web system. Broken object level authorization (BOLA) is the most common API threat, represented in about 40% of all API attacks.This is due to the common practice of using object identifiers to retrieve and manipulate data via API endpoints. For example, in the URL example.com/profile/1, the 1 is an identifier, which represents the id of the profile to return.
Broken User Authentication ●	Incorrectly implemented API authentication may allow attackers to impersonate API users and access confidential data. Broken user authentication enables attackers to use stolen authentication tokens, credential stuffing and brute-force attacks to gain unauthorized access to applications. This can manifest in direct browsing and many access control issues. Failure to effectively authenticate users presents a large threat to the API providers and also the users whose data resides within the API.

Excessive Data Exposure	Many APIs err on the side of exposing data and count on the API user to filter the data properly. When APIs provide more data than is needed, an attacker can exploit an app by using redundant data to further extract sensitive data such as personal email addresses and other personally identifiable information The API may expect the API client to filter out such data so that it's not presented to the end user, but this does not prevent it from being read during transport, exploited by a malicious API client, or even inadvertently revealed by a benign API client.
Broken Function Level Authorization	Improperly implemented user authorization allows unauthorized users to execute administrative API functions such as adding, updating or deleting a customer record or a user role. Broken Function Level Authorization (BFLA) is similar to BOLA, but is focused on general functions rather than individual objects.
Mass Assignment	The API automatically applies user inputs to multiple properties. An attacker could use this vulnerability to, for example, change themselves to an admin while updating some other innocuous property of their user profile. APIs that directly consume input requests and assign/write them to the business logic data stores are vulnerable to mass assignment. In the case of an object database, for example, if the payload maps directly to the stored data and is inserted directly, without input validation compared against authorization levels, then the attacker can alter data in unintended ways.
M&A Risk Analysis	Discover and analyze vulnerabilities of a potential acquiree before it becomes a threat to the acquirer
Security Misconfiguration	This covers a variety of API configuration mistakes, including misconfigured HTTP headers, unnecessary HTTP methods and what OWASP calls "verbose error messages containing sensitive information." Security misconfiguration is a catch-all for a wide range of security misconfigurations that often negatively impact API security as a whole and introduce vulnerabilities inadvertently. Significantly, a service's default configuration can be a Security Misconfiguration. Any information on or insight into a service may give an attacker valuable information that can be used for exploitation. For example, HTTP servers may present version numbers in default response headers or bodies. Such information can influence an attacker's plan. Non-verbose responses like custom 200 responses should be considered to accommodate a true defense in depth posture.
Injection	The attacker sends specialized commands to the API that trick it into revealing data or executing some other unexpected action. This attack is the one hold-over from the original OWASP Top 10 list – the others are new and focused just on APIs. Attackers exploit injection vulnerabilities by sending malicious data to an API that is in turn processed by an interpreter or parsed by the application server and passed to some integrated service. The injection vulnerability is caused by not validating user input, where that input is later used verbatim without any protection mechanisms. For example,the input, if used as an update to a field in a relational database, may contain text that terminates the SQL query and performs additional queries. If the input is not sanitized to remove potential SQL query string modifications, then it will be executed as a successful statement.

Improper Assets Management **NOT TESTED** ●	Attackers are able to find production APIs and deprecated APIs. An outdated or incomplete inventory results in unknown gaps in the API attack surface and makes it difficult to identify older versions of APIs that should be decommissioned. This can be caused by the API not being properly documented, API developers turning over in an organization or other negligence. Since Synack works from documentation provided by the client, testing for extra assets is considered out of scope.
Lack of Resources & Rate Limiting **NOT TESTED** ●	APIs that improperly implement rate limiting or neglect to implement it at all are highly susceptible to brute-force attacks or DoS. APIs that don't have restrictions in place can be overwhelmed by legitimate requests, as well as requests from malicious actors. In such situations, an API can no longer operate and will no longer be able to service requests and can be unable to complete those currently in progress. Testing for this vulnerability results in a significant possibility of Denial of Service (DoS) or significant degradation of customer assets. Due to this risk, Synack does not test this flaw.
Insufficient Logging & Monitoring **NOT TESTED** ●	Insufficient logging and monitoring, combined with missing or ineffective integration with incident response, allows attackers to perform reconnaissance, exploit or abuse APIs, compromise systems, maintain persistence, advance attacks, and move laterally across environments without being detected. OWASP notes that studies show it typically takes over 200 days to detect a breach. Detailed event logging and close monitoring may enable API developers to detect and stop breaches far earlier. Insufficient Logging & Monitoring is not a direct vulnerability or threat, but rather the organization is blind to current active attacks, previous attacks and the information needed in the forensics process to determine the impact of the attack. Without this insight, the organization is vulnerable to future attacks through the same methods or backdoors planted in previous attacks, which might be even more difficult to detect. This logging and monitoring capability is internal and should be planned from the initial design stages of projects. This can be tested to see if the system has logs during a Synack test. For example, does your team see our testing and have logs to show when and where we were at? That said, from our SRT Researcher perspective, they are normally unable to see the logs and determine if they are capturing the traffic. Additionally, the term "insufficient" is impossible to determine without further business analysis. For example, in some systems a week of logs is sufficient and possibly preferable due to legal liabilities. For others, a year or two of logs may be preferable. There are legal, business, cost and framework requirements that go into this decision. This is a business-driven decision that the customer team should evaluate and decide the risk analysis, cost and benefits of a solution and the rules around it. To test the system though, clients can simply check their logs to see if they caught the testing traffic. Synack does not directly address this control due to our external black-box testing methodologies.

Other types of API testing (what Synack does not provide)

Validation testing

Validation Testing addresses questions such as: Was the correct product built? Is the designed API the correct product for the issue it attempts to resolve? Was there any major code bloat—production of code that is unnecessarily long, slow and wasteful—throughout development that would push the API in an unsustainable direction?

The second set of questions focuses on the API's behavior: Is the correct data being accessed in the predefined manner? Is too much data being accessed? Is the API storing the data correctly given the data set's specific integrity and confidentiality requirements?

The third set of questions looks at the efficiency of the API: Is this API the most efficient and accurate method of performing a task? Can any codebase be altered or entirely removed to reduce impairments and improve overall service?

These decisions are based on the design and business needs of the customer.

QA testing

These tests ensure the API performs exactly as it is supposed to. This test analyzes specific functions within the codebase to guarantee that the API functions within its expected parameters and can handle errors when the results are outside the designated parameters.

This is essentially a "smoke test" for the engineering team. This ensures that the APIs are doing what they planned. This is something that should have been done before testing with Synack.

Load testing

This is used to see how many calls an API can handle. This test is often performed after a specific unit, or the entire codebase, has been completed to determine whether the theoretical solution can also work as a practical solution when acting under a given load. This load can be either the operational average load to check performance or an expected maximum peak load as determined by the product development team.

This is a test that should be completed by the design and engineering team of the customer to ensure we are able to test at scale. Although we can create significant traffic, Synack does not provide this type of testing. A failed test from this will degrade the performance of the system, possibly causing a crash or shutdown. This would effectively test denial of service, which is not a product we offer due to the inherent risks.

Reliability testing

Reliability tests ensure the API can produce consistent results and the connection between platforms is constant. Obviously, if Synack is testing an API and results are not stable or consistent, there is little we can determine. This is also something that is normally done during development and not during security testing. If this testing is not done and the system produces inconsistent results from identical input, Synack's security testing will be less effective.

Fuzz testing

Fuzz testing forcibly inputs huge amounts of random data,also called noise or fuzz,into the system, attempting to create negative behavior, such as a forced crash or overflow. If you read the words "forced crash," "negative behavior" and "overflow" and got a little worried, good. Those are destructive and disruptive behaviors in a system. These are effectively creating a denial of service crash state on the tested system. This is testing that Synack does not work on due to the risk of adverse effects on the customer assets.