Guide to API Security Testing

Written by Jeff Forristal

Information Security Industry Expert

Guide to API Security Testing

Containerization and microservices are no longer just hype. While exact adoption rates are debatable[1], both have become foundational tools used by organizations looking to publish APIs for internal or external consumption. Microservices are generally defined as a service-oriented architecture (SOA) variation[2]; however, there is no formal protocol definition (unlike the historical SOAs that leverage clearly defined protocols including SOAP, WSDL, etc.). The lack of a clear protocol makes application security assessments of microservice APIs somewhat precarious, since the typical go-to web security assessment tools, prescribed security assessment methodologies, and general penetration tester experience may not include coverage or interaction know-how for a particular microservice API offering or operational behavior. Public microservice APIs are often exposed for direct interaction in B2B and mobile application support scenarios, meaning their risk profiles slightly differ from typical web usages and the related catalog of top security problems à la OWASP Top 10[3].

Testing Challenges: API Discovery

The first challenge of microservice API testing is simply finding, or discovering, the APIs to assess. APIs are often decoupled from web properties, meaning typical web assessment tools like web crawlers and web browsing proxies will be ineffective. Forceful browsing and other brute-force URL discovery methods leveraged against a known API host/endpoint may yield some initial results, but often such approaches will not divulge the API's parameters and thus lead to inefficient assessment coverage. Better would be to import a structured definition of the APIs into a security assessment tool, à la how WSDLs can be imported for automated SOAP assessments. However, the lack of a core protocol definition for microservices has caused multiple different API definition formats, e.g. Swagger, OAS, RAML, API Blueprint, and WADL. Quite often security tools lack the ability to handle some (or all) of those formats, ultimately leaving API security assessors left with manual targeting and entry point definition as part of the initial phase of configuring/aiming the security testing tools. Public B2B APIs are often, by nature, documented in a manner that should be sufficient for

Microservice API definitions can come in many forms:

- Swagger
- OAS
- RAML
- API Blueprint
- WADL

1 https://containerjournal.com/2017/01/16/measuring-docker-adoption-rates-requires-precision/

2 https://en.wikipedia.org/wiki/Microservices

3 https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

security assessors. Private mobile application support service APIs, however, may not be publicly documented outside the interactions performed by the (binary) mobile application—meaning security assessors should be prepared to leverage mobile application reverse engineering or network man-in-the-middle techniques to divulge API existence/details.

Testing Challenges: API Interaction

Even when a security assessor takes the time to manually target a web security tool at a known microservice API endpoint, interoperability is still a significant challenge. Authentication and authorization operation widely vary, leveraging everything from static values (API keys) to dynamic tokens (JWT, OAuth), provided in HTTP request headers or query/path parameters. Some implementations may use mutual SSL authentication, necessitating a client SSL certificate. Your web security assessment tools must be flexible enough to accommodate a wide variety of authentication approaches and dynamic token refresh procedures, including the potential for custom-coded logic if need be.

Be prepared to interact with microservice APIs in various ways:

- **REST**
- **Thrift**
- **Avro**
- **HTTP/2**
- **gRpc**
- **Protobuf**
- **JSON**

In fact, the need for custom-coded logic doesn't end there. API service results are often designed for specific-purpose programmatic consumption, necessitating specialized interpretation or scripted tool tailored to contextually process the results correctly. The result data format itself can vary beyond typical JSON, including binary formats such as Google protobuf. Even the network/framework transport can vary beyond the popular HTTP/1.x REST, necessitating specific clients for things like gRpc, Thrift, Avro, HTTP/2, etc. Overall, a security assessor should have the capability to adapt existing tools/scripts and even create new security assessment scripts for use, to gain the best testing coverage. Methodologies may also need to be appropriately adjusted, to alleviate any tool-particular operation or HTTP/web assumptions that do not apply in microservice API assessment scenarios.

Mobile application assessments may require additional reverse engineering or network man-in-the-middle interception to deduce API interaction with supporting backend API/services; in some cases, specialized tools may be necessary to reconstruct API data structures (e.g. protobuf) from binary or decompiled mobile

application code. Recent mobile OS-induced HTTPS/SSL traffic requirements (such as IOS App Transport Security[4]) and growth of SSL pinning[5] implementations further requires assessment methodologies to include SSL pinning circumvention techniques as part of their standard practice, simply to witness API interaction particulars.

Testing Practices: Microservice API Attack Considerations

Microservice APIs are just software, and therefore can generally be affected by any type of common software flaw such as those found in the Common Weakness Enumeration (CWE) list[6]. Beyond the traditional top risks outlined in the OWASP Top 10[7], security assessors of microservice, B2B, and mobile APIs should additionally assess risk in the following API-particular areas.

Application DDoS/Processing Avalanche

Netflix made the news in 2017 when they released a blog article[8] along with a DEFCON 25 presentation detailing the impact of application DDoS attacks[9] against microservice architectures, which they proved by conducting an intentional and successful application DDoS attack against their own infrastructure. The blog article describes the attack as a "focus on expensive API calls, using their complex interconnected relationships to cause the system to attack itself," effectively causing a processing avalanche that is asymmetrically inexpensive for an attacker to trigger but costly for the system to handle. The risk is increased if such processing is accessible without authentication, allowing an anonymous attacker to drastically impact the system without accountability.

The Netflix blog article includes a recommended testing methodology and showcases the release of various testing tools, which can be adapted for inclusion into a security assessor's methodology for microservice APIs. Security assessors should analyze the DDoS potential of the processing cost/overhead in handling API calls for any public B2B, mobile, or microservice invocation, to ensure the assessed backend APIs cannot be intentionally crippled with minimal effort.

---

4 https://developer.apple.com/news/?id=12212016b

5 https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning

6 https://cwe.mitre.org/

7 https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

8 https://medium.com/netflix-techblog/starting-the-avalanche-640e69b14a06

9 https://en.wikipedia.org/wiki/Denial-of-service_attack

Rate Limiting/Throttling Abuses

Microservice APIs are often provided as part of B2B/commercial API services (i.e. SaaS), potentially with varying service levels for different customers. It is common to involve customer rate limiting or throttling mechanisms as part of an B2B API implementation, whereby certain customers/service levels are only allowed to consume a limited amount of the service compared to higher tier customers/service levels. Consumption can be measured in varying ways, but often it's done by counting the number of service invocations/requests.

Bypassing or abusing the rate limiting/throttling mechanism results in a theft of services risk that is not only a loss of revenue (since the customer is not paying for the extra services), but may also have negative operational cost impact (since the API provider is still paying for the processing of those rendered services). API security assessments should have a methodology and supporting test tools that can determine if an API endpoint has implemented a rate limiting mechanism, how that limiting mechanism aggregates requests towards the limit (e.g. global requests account, by IP address, by session token, by API key, etc.), and methods to bypass the limit mechanism (e.g. by switching IP addresses, by refreshing the session token, etc.). If rate limits are sufficiently enforced per caller, then the security assessor should evaluate methods to create/register multiple caller identities to use consecutively for aggregate rate limiting bypass.

Custom Authentication & Authorization Mechanisms

Microservice APIs can use conventional authentication/authorization patterns; however, the mechanisms are often custom implemented and thus prone to typical coding errors. The OWASP Top 10 2013 list already includes the "A2-Broken Authentication and Session Management" risk, although many assessment methodologies emphasis testing of session cookies and various session state tracking mechanisms utilized by web service frameworks – mechanisms that are generally not present for microservice APIs.

When assessing an API, a security assessor should first focus on becoming intimate with the intended workings of the authentication/authorization leveraged for the target implementation. Once the primary mechanism has been identified, the assessment methodology should include tests for weaknesses. For example:

- Foregoing or duplicating HTTP headers that carry auth values/tokens
- Performing injection attacks for the API auth value/token
- Noting if the API auth value/token is provided as a GET parameter, potentially exposing the value/token in HTTP request logs
- Testing for inappropriate or weak uses of cryptography and randomness within the auth mechanism
- Checking if auth enforcement is applied inconsistently, i.e. forgotten on certain API endpoints or certain requests subtypes/verbs of the same API endpoint

Binary Deserialization/Parsing Attacks

Various binary formats may be leveraged as part of a microservice API implementation: Avro, Thrift, protobuf, etc. Incoming data in these formats needs to be deserialized or parsed into a logical object for the implementation to operation on. A historical review of CWE shows parsers and deserializers are notorious for having bugs in the parsing of incoming, potentially malicious data. The implications can range from denial of service attacks (excessive CPU consumption) to remote code execution. A recent (September 2017) example of this is a remote code execution vulnerability in the Apache Struts deserializer of RESTful requests[10] (CVE-2017-9805).

An experienced security assessor should take time to review all data formats received by a microservice API, and consider if any parser or deserialization attack testing is warranted. A mature assessment methodology should include tools to derive a corpus of corrupt or malicious inputs that can be sent to an API service as part of security testing. Reverse engineering of API clients, particularly mobile apps, may be necessary to deduce the serialization formats being used by an API implementation.

---

[10] https://nakedsecurity.sophos.com/2017/09/06/apache-struts-serialisation-vulnerability-what-you-need-to-know/

What You Can Do

The best place to start is to take a serious look at your current security assessment capabilities, to determine if your methodologies or tooling have gaps, specifically in handling microservice, B2B, and mobile API assessments. Be sure to inquire with your third-party security assessors regarding how they approach APIs differently than a typical web security assessment. Security program managers should realize that standard web security assessment tools may not be sufficient for testing microservice APIs, and should allow some budget for security teams to review or build tools to fill API assessment gaps.

You should also encourage your security team members to learn/grow in DevSecOps topics, so they remain exposed to emerging and standardizing technologies relating to microservices, APIs, API implementation patterns, and common API frameworks (along with the offered security capabilities/features). That learning can be extended to partnering with development teams and augmenting their existing Continuous Integration (CI) tests and testing frameworks to additionally perform security service tests, alleviating the security team from having to deploy a second API testing framework for security-exclusive test content.

Lastly, all organizations developing APIs should have clearly identified and preferred strategies or technical architecture/design patterns to address the common API attacks prior discussed, to ensure defensive consistency. Microservice developers should routinely include circuit breaker, load shedding, and work timeout/abort patterns as standard for any API implementation, while security teams assess the operation of these implementations. Custom authentication and authorization mechanisms are a critical review area; those mechanisms would benefit from a deep source code review in additional to typical penetration testing/assessments.

Jeff Forristal

Jeff is a security technology professional with over 18 years of experience in the security industry. Throughout his professional career he has been responsible for conceiving new security service offerings, developing industry-first and market-leading product features, educating customers on security operations, and driving research into new security industry areas.

He has written multiple features and cover-story articles for Network Computing and Secure Enterprise magazines. Under the pseudonym “Rain Forest Puppy,” he has been recognized as an industry expert in web application security and was responsible for industry landmarks including the first documented discovery of SQL injection, the first responsible security disclosure policy, and the first intelligent web application scanner.