



Creating a Safe Environment for Under-Protected APIs

WHITE PAPER



TABLE OF CONTENTS

▶ Background	3
What are APIs?	3
▶ APIs in Modern Applications	4
Rapid FaaS Evolution is Driving API Adoption.....	5
▶ Security Concerns of Distributed API Architecture	5
The API Security Visibility Problem	5
API Security Risks and Vulnerabilities	6
▶ Designing a Secured API Environment	8
Inline API Security Measures	8
Radware API Security Technology	9
▶ References.....	10

➔ Background

As the popularity of applications continues to grow, the adoption of application programming interfaces (APIs) is increasing. APIs enable applications to interoperate with other services by integrating different clients and applications across multiple services. They save time and facilitate the flexible development of microservices architecture apps, agile development methodologies and continuous delivery.

While APIs bring tremendous benefits, they also introduce new security risks, including service disruption and data theft. Many APIs process sensitive personally identifiable information (PII). Additionally, known application security risks with HTTP/S apps are as relevant for APIs as they are for web applications. Communication with APIs usually follows known structures and protocols. The most common protocol is REST/JSON, which has a schema format definition called OpenAPI.

Because threats vary, API security requires a combination of access controls (such as authentication and authorization mechanisms), injection prevention, bot management and DoS mitigation. In addition, hackers may try some API-specific attacks such as using invalid schemas, parameter tampering or token manipulations. So, in addition to support for OpenAPI, API security should also address unknown, undocumented APIs.

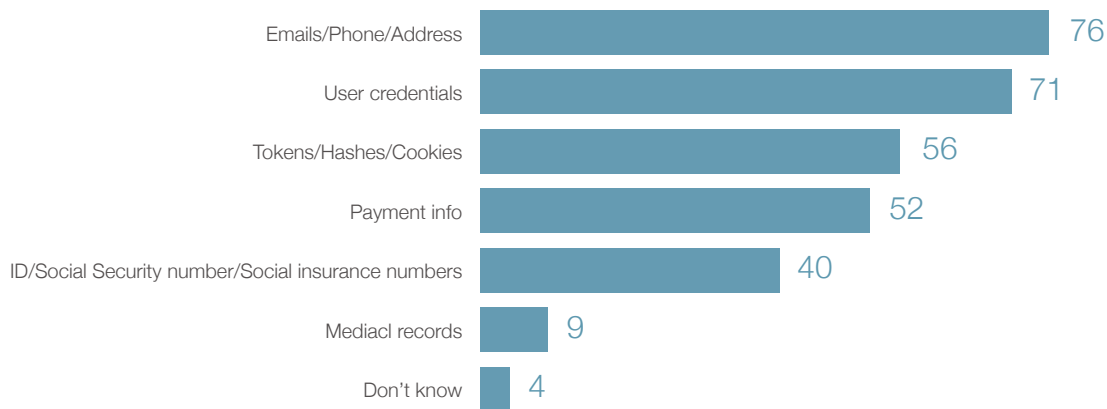


Figure 1: Examples of various operations comprising the application delivery service

What are APIs?

APIs are a set of tools and protocols used to develop application software. These interfaces incorporate a predefined request–response message system that exposes reliable content and operation negotiation. The most common API protocol in modern architectures is REST. Other protocols have recently emerged— such as gRPC — for example, to support developments and the delivery of the Kubernetes architecture.

Documented vs. Undocumented APIs

APIs can be divided into two types:

Documented APIs — where the API provider includes documentation on how to use the APIs through a definition language. Currently, the most common definition language is Open API Specification (OAS), also referred to as Swagger (though this is the name of a tool rather than a format). Other formats such as SOAP and RAML are also available, but their adoption is significantly lower and declining.

▶ **Undocumented APIs** – How to use the APIs is not formally documented in a definition language.

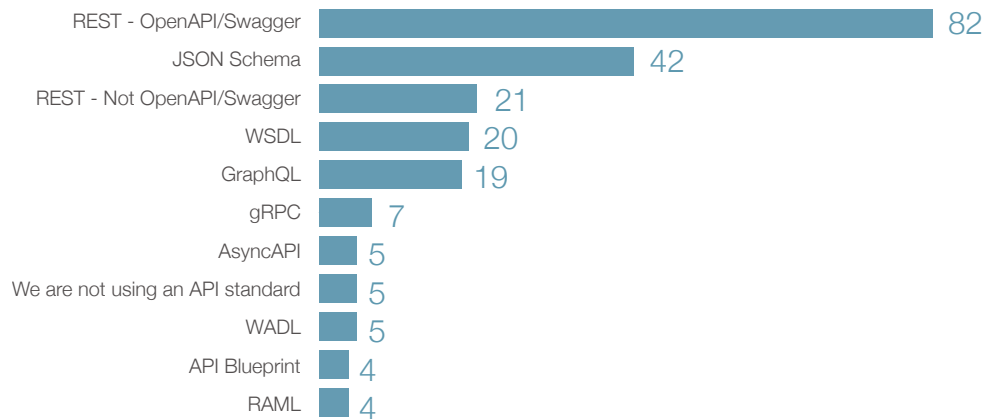


Figure 2 - Common standards for defining APIs, The State of API 2020, SMARTBEAR (n=3,536)

➔ APIs in Modern Applications

APIs are used in a variety of modern applications and the number of use cases is continuously growing. The most common examples are:

- ▶ Web APIs, mostly in single-page applications
- ▶ Mobile applications
- ▶ Embedding public and third-party APIs as external services into an existing application (e.g., Google Maps APIs).

DevOps environments — with the ever-increasing demand for continuous delivery — require complete process automation utilizing APIs across the board:

- ▶ Service provisioning and management (e.g., AWS API)
- ▶ Platform management apps
- ▶ Continuous delivery process automation

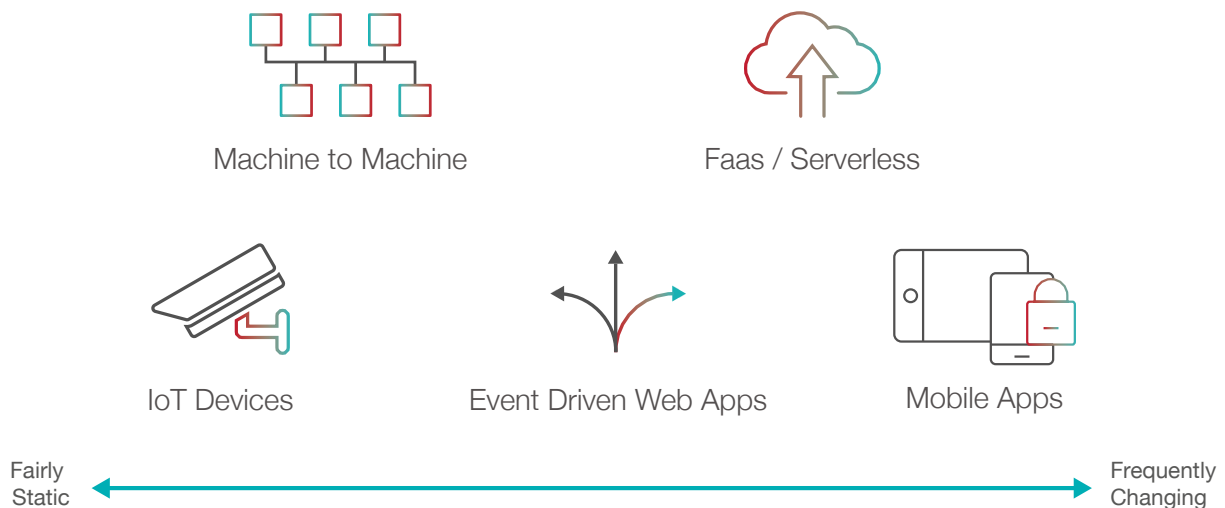


Figure 3 - The API Economy and use cases

Rapid FaaS Evolution is Driving API Adoption

Serverless architecture — or FaaS (Functions as a Service) — offers a model where the operational unit is a set of functions rather than a web server. These functions may be internal for East-West interfaces or an API exposed to the client-side North-South interfaces, which may invoke these APIs upon relevant client-side events.

In a FaaS architecture, the management of functions is greater in complexity than just managing lasting virtual machines. The function containers are created upon request and may disappear immediately after being used. This approach simplifies the development process and dramatically reduces operating expenses (OPEX). It is important to state that APIs are not tightly coupled with FaaS and are widely used in other architectures and with web applications.

➔ Security Concerns of Distributed API Architecture

The API Security Visibility Problem

API vulnerabilities are hard to monitor and do not stand out. Traditional application security assessment tools do not work well with APIs or are simply irrelevant. For example, Dynamic Application Security Testing (DAST) and application scanning tools cannot invoke the API because they cannot generate well-formed requests, even if the tools know whether the request body should be a JSON or an XML.

Similarly, Static Application Security Testing (SAST) tools are limited in their ability to scan API code, as is typical in an API. Instead, third-party frameworks and libraries use custom methods to read a JSON or XML document from the body of the HTTP request, then parse it and pass the data into the API code. These methods are different from one another and are subject to changes, limiting the success rate of static tools.

When planning for an API security infrastructure, authentication and authorization must be taken into account, yet these are often not addressed properly in many API security solutions.

“Since Application Programming Interfaces are mission critical and involve crucial business functionalities and processes, API security has become a major concern and challenge for organizations.”

- Open Web Application Security Project® (OWASP) Top 10 Web Application Security Threats, 2019

AUTOMATION	DISCOVER & MONITOR	VISIBILITY & SUPERVISION
<ul style="list-style-type: none">• API to Manage Policies• Import and Enforce OpenAPI• Import and Enforce API Catalogue• Extract Secured APIs	<ul style="list-style-type: none">• API Discovery• Categorization• Analysis	<ul style="list-style-type: none">• Logo of Violations• Attack Reports• Per-API access count• Traffic Volume

Figure 3 – Operational challenges in API security

API Security Risks and Vulnerabilities

APIs are vulnerable to all types of attacks and threats against web applications. Most of the APIs are REST APIs with JSON bodies (REST-JSON), which run on top of the HTTP protocol. As such, most of the web application security risks are just as relevant for APIs. Additionally, APIs introduce other security challenges mostly around access control as the APIs may be served independently and not only as a whole set of web application resources.

APIs submit and retrieve data and may expose application logic and potentially sensitive data. As a result, they have increasingly become a target for attackers trying to find easier ways into applications.

THREAT	RISK	PROTECTIONS/CONTROLS
Bot attacks	<ul style="list-style-type: none"> Account Takeover (credential stuffing/ cracking) Scraping/data exfiltration Fraud 	<ul style="list-style-type: none"> Apply dedicated machine learning on API calls as well as HTTP traffic to detect suspicious behaviors of sophisticated bots Quota Management - Limiting the number of calls allowed per source for each API. It is an important protection against service abuse (for informational APIs), ATO attacks and denial-of-service (DOS) attacks
API transaction manipulations	Confidentiality and integrity of data in transit	TLS is required to secure the communications between the client and APIs for transport confidentiality and integrity of data in transit
TCP protocol attacks and evasion techniques	TCP packet replay, TCP packet fragmentation and TCP packet reordering	Once the evasion attempt or the protocol manipulation is detected, an immediate TCP Termination should be initiated
HTTP protocol attacks and evasion techniques	Manipulation of HTTP headers (for instance, a content-type header that is not aligned with the content sent in the body, etc.)	<ul style="list-style-type: none"> HTTP protocol parsing and enforcement of HTTP RFC protects against various HTTP attacks such as NULL byte injection, encoded attacks, HRS attacks, content-type mismatch, etc. Traffic normalization for evasion attacks detection. Peacetime patterns should be used as a reference as encoded attacks can easily bypass security solutions Message size policy enforcement on HTTP messages, body, headers and JSON/XML element sizes secures the application against buffer overflow attacks, resource exhaustion and other availability attacks on API infrastructure
POSTed JSONs and XMLs injections	May eventually reach databases, leading to injections	<ul style="list-style-type: none"> Strong Typing and the Positive Security model provide tight protection of API infrastructures. It is impossible to generate most of the attacks if, for example, the only allowed value type in the JSON element is an integer with the value range of 1–100 XML/JSON validity check and Schema Validation is extremely important security protection. Types, value ranges, sizes and order of XML elements must be configurable SQL and no-SQL injection protections through sanitizing and validating user inputs and rule-based attack detection

THREAT	RISK	PROTECTIONS/CONTROLS
Insecure direct object references	Manipulation of state information in parameter value that stores the account number may allow access to unexpected data	<ul style="list-style-type: none"> • Session and field protections against manipulation. • Input validation in post transactions to detect injections of references
Unvalidated redirects	External entity embedding malicious content in the service or application	Validation of user inputs for external domains in parameters and submitted forms and form fields
Data leakage	Credit cards, social security numbers, passwords or other sensitive data may leak through the API response and 500 error messages may leak architecture information exposing server and data-storing types	Data leak protection ensures error messages and sensitive information is not leaking to the potential attacker. Data structures and schemes of private information should be recognized and guarded
Access violations and abuse of APIs	Unexpected users may invoke APIs they should not have access to or perform operations they should not be allowed (e.g., delete a license vs. just generating a license)	<p>Access Control policy management with:</p> <ul style="list-style-type: none"> • IP-based and geo-location restrictions when relevant • Access restriction to APIs where, for example, should expose some APIs for public access while others are just for internal use • Access restrictions to specific HTTP methods where the set of operations that are allowed for some users is restricted for others. For example, a user can generate a license but cannot delete the license once generated
DOM XSS	An API client-side vulnerability that may lead to additional attack vectors	XSS protection based on rules and signatures of known attack patterns. These rules and signatures must be continuously reviewed and updated
Session management attack	Session hijacking and privilege escalation attacks	Session management protection of the API key, which is posted as a body argument or in the cookie
Buffer overflow and XML bombs	Large JSON/XML element values can affect performance, resource consumption and service availability	<ul style="list-style-type: none"> • Strong typing and a positive security model provide tight protection to API infrastructure. It will be impossible to generate most of the attacks if the only allowed value type in the JSON element is an integer with the value range of 1 – 100 • XML/JSON validity check and schema validation is extremely important security protection. Types, value ranges, sizes and order of XML elements must be configurable
DDoS attacks	<ul style="list-style-type: none"> • Network and application-based DDoS attacks • Distributed Denial of Service (DDoS) 	<p>DDoS protection – Attacks against applications can come en masse, looking to disrupt services or retrieve database access. Some attacks focus on rendering the web application layer unreachable, causing a denial-of-service state. One HTTP request can lead the server to execute a large number of internal requests. Multiple requests can consume all its resources. There are few techniques to do that, including randomizing URLs, API keys and bypassing the application caching.</p>

Designing a Secured API Environment

Inline API Security Measures

Web application firewalls (WAFs) and API gateways are the two primary inline security tools for API protections. While API gateways usually offer authentication and authorization features, their HTTP traffic and payload analysis, as well as their OWASP Top 10 API security risks and web protection offering is either limited or absent.

On the other hand, most WAFs do not understand the differences between APIs and regular web applications, even when both may use the HTTP RFC as a baseline. The obvious example is the HTTP method used in REST APIs. While it is extremely unlikely for a web application to accept HTTP requests with the DELETE HTTP method, it is a common practice with REST API call to support this method and other potentially risky methods for web applications such as PUT.

It is imperative that security measures be applied to enforce security policy on documented and undocumented APIs.

Suspicious Behaviors and Violation Indicators

To secure APIs, a security solution must first discover the undocumented APIs and use OpenAPI/Swagger schema definitions for documented APIs to enumerate the API endpoints (referred to as “paths” in OpenAPI) to be secured and then enforce the appropriate measures, blocking API calls that attempt any of the following violation indicators:

- ▶ Accessing restricted APIs
- ▶ Using non-defined/non-allowed HTTP methods for an API endpoint
- ▶ Containing unrecognized and non-valid parameters
- ▶ Containing out-of-expected-range parameter values
- ▶ Embedding web attacks in JSON payloads or parameters
- ▶ Excessively utilizing the APIs
- ▶ Extracting sensitive data through the API
- ▶ Attempting to take advantage of API vulnerabilities
- ▶ Attempting to break the API authentication process through an account takeover (ATO) attack

Most Common Attacks Against APIs

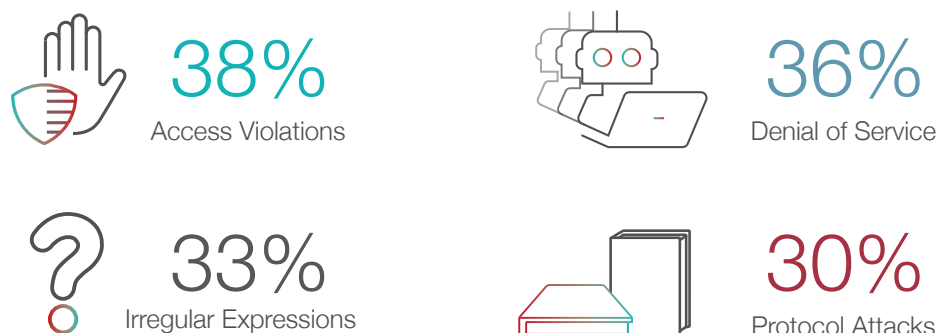


Figure 4: Most common attacks against APIs – Radware Application Security Study

Radware API Security Technology


By combining positive and negative security models, Radware secures APIs from known and zero-day attacks as part of its flexible and scalable web application security solution and is recommended by NSS Labs and ICASA Labs across on-premise, cloud, virtual, stand-alone or integrated and inline and out-of-path deployments.

- ▶ **Positive security model** defines the allowed actions while blocking all access attempts to non-listed API endpoints/paths. API catalog and schema validation are great examples of a positive security model. The value of such an approach is a tighter, more effective security policy. The benefit of using an OpenAPI for security is the ability to define a positive security model immediately, without any learning process to effectively secure the APIs.
- ▶ **Negative security model** defines the prohibited actions based on signatures and rules of known types of attacks, while API calls without a match to the negative model are allowed. The value of such an approach is not being dependent on an API structure and protecting against known web and API vulnerabilities and weaknesses.

Beyond all the required protections discussed above, including SQL injection, broken authentication, XSS, CSRF, DDoS, etc., Radware's web application security technology features additional attack correlation capabilities, which allows blocking of repetitive attack sources by managing a penalty score for security violations per source. Once an attack source reaches a predefined score threshold, it will be blocked.

Radware introduces a unique auto policy generation mechanism to reduce the complexity of keeping evolving environments secure. Advanced machine learning algorithms learn XML, JSON structures and schemas are used for enforcement as part of the validation phase and create a security policy based on results. Moreover, these algorithms are able to track changes in the application and perform automatic updates in real-time, producing an adaptive security model.

Radware Bot Manager features purpose-built, intent-based deep behavioral analysis (IDBA) of bot traffic to secure APIs from bad bots attempting to break into user accounts and steal sensitive information. Among various factors, the detection engine uses a deterministic rule engine, integrity checks and collective intelligence to understand the API invocation context and flow control (including authentication flow), thus preventing data leakage due to account takeover attacks. It also features a software developers kit (SDK) with innovative unique source identification capability to fingerprint to machine communication.



Questions: Tell us more or give us a call at
877.524.1419 to reach a sales professional.

About Radware

Radware® (NASDAQ: RDWR) is a global leader of [cybersecurity](#) and [application delivery](#) solutions for physical, cloud and software-defined data centers. Its award-winning solutions portfolio secures the digital experience by providing infrastructure, application and corporate IT protection and availability services to enterprises globally. Radware's solutions empower more than 12,500 enterprise and carrier customers worldwide to adapt quickly to market challenges, maintain business continuity and achieve maximum productivity while keeping costs down. For more information, please visit www.radware.com.

Radware encourages you to join our community and follow us on: [Radware Blog](#), [LinkedIn](#), [Facebook](#), [Twitter](#), [SlideShare](#), [YouTube](#), [Radware Connect](#) app for iPhone® and our security center DDoSWarriors.com that provides a comprehensive analysis of DDoS attack tools, trends and threats.

© 2021 Radware Ltd. All rights reserved. The Radware products and solutions mentioned in this ebook are protected by trademarks, patents and pending patent applications of Radware in the U.S. and other countries. For more details, please see: <https://www.radware.com/LegalNotice/>. All other trademarks and names are property of their respective owners.