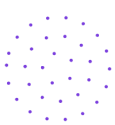
SALT

API Security Evaluation Guide

WHITE PAPER

API Security Evaluation Guide

TABLE OF CONTENTS



Modern applications are built on APIs, and application security practices now heavily depend on API security practices. As an industry, the recognized approaches and tooling for securing traditional applications are fairly well understood. However, mitigating API attacks differs due to a shift in where application logic resides as well as a de-coupling and de-emphasis on client front ends. Securing APIs requires consideration of many security domains including network, infrastructure, IAM, data, and not just application code. Organizations must evolve their API security strategy to protect their business, their users, and their data.

With API attacks on the rise, and existing security technology proving to be ineffective at stopping API attacks, organizations need to take a new approach. Salt Security is providing this research to industry to improve awareness of what it takes to adequately secure APIs and how to evaluate a given API security offering. We also want to enable decision makers within organizations to vet vendor claims and map to security functionality that is necessary to protect their business.

As with any new technology, organizations don't always know how to evaluate “good” and measure what features may be beneficial. You can use the information contained in this evaluation guide to help assess the quality of a respective vendor offering. We’ve organized the criteria into the following sections:

- Design and architecture
- API and sensitive data discovery
- API attack detection
- API attack prevention and blocking
- API-centric incident response
- API vulnerability identification and remediation insights

We start with a brief overview on how the attack surface has changed and why a new approach is needed. Then each section provides the key aspects of that functional area and a list of critical functions to look for in a given offering. The guide wraps up with a conclusion and a brief overview of the Salt Security platform approach. We hope the information helps you evaluate API security platforms.

Why do we need more than application security, and why now?

Traditional application security approaches and tooling capture only a small portion of the range of potential API attack types, and they provide limited efficacy in

detecting or blocking such attacks. Organizations hit a wall when attempting to integrate and operationalize traditional security tooling and process at scale, which inevitably leaves their APIs at risk.

Complicating factors for the API landscape include:

- **Acceleration of API creation** to facilitate business and exchange data with customers as well as partner ecosystems
- **Rapid development and release** of APIs supported by agile development methodologies and DevOps practices
- **Higher volume of APIs and distribution** of them as a result of microservices architecture and cloud-native design patterns
- **Evolution from XML and SOAP** APIs which were largely internal to REST, GraphQL, and gRPC that are more publicly exposed
- **Organization-specific API architecture** and business logic implementation
- **Advancing attacker methodologies** where attackers circumvent access controls and abuse business logic, not just exploit vulnerabilities in code or deny service.
- **Multiple front-end client types** that make endpoint and client-side controls untenable
- **Increasing privacy regulation** governing collection of sensitive data and exposure of PII

To address modern threats that exploit flaws in APIs, organizations are looking for a range of new capabilities – prevalent API attacks have revealed a particular need to identify:

- Sensitive data exposures that impact privacy
- Broken object level authorization (BOLA) attacks that result in privilege escalation
- Credential stuffing that results in account takeover (ATO)
- Enumeration and scraping attacks that lead to mass data leakage

Organizations today support more APIs of various types and protocols than ever, all with varied levels of exposure. As a result, organizations face a massive, shifting attack surface. Traditional security approaches and mechanisms aren't enough – we need a new approach.

What should organizations look for in an API security offering?

Any API security offering you consider should be built as a platform of capabilities and not a one-off tool or scanner. API security strategy demands a full lifecycle approach since security issues, vulnerabilities, logic flaws, misconfigurations, and more arise at different stages of design, development, delivery, and operation.

Architecture matters. Any platform you are considering should leverage big data to collect and store large amounts of API telemetry, correlate API traffic, provide context, and power fast attack detection and response. The platform should also use AI/ML so that it can continuously extract useful, actionable signals for development, operations, and security teams. Time-in-market is another key consideration – since algorithms improve over time through training, and data sets are enriched by the network effect, with more users and API calls. When architected properly, such aggregated, anonymized data benefits all consumers of the platform while still preserving privacy.

Based on Salt customer requirements, collective experience, and industry security best practices, we've defined the following design criteria and capability areas as critical for any API security offering that an organization may be considering:

- Design and architecture
- API and sensitive data discovery
- API attack detection
- API attack prevention and blocking
- API-centric incident response
- API vulnerability identification and remediation

Subsequent sections detail the core capabilities that organizations should look for when evaluating a respective vendor offering. Criteria are numbered so they are easily referenceable, but this does not infer priority. Organizations have varying sets of business and security requirements, and some API security capabilities may be more or less important.

Design and architecture

Design and architecture of the API security offering should:

- Support heterogeneous environments and secure all types of APIs anywhere
- Avoid the use of additional client-side code, server agents, or network proxies

- Leverage cloud-scale data storage and analytics
- Use AI/ML to continuously analyze the organization's unique API business logic

Any API security offering you consider should be built with automation and cloud-scale capacity in mind. Realistically, this infers a cloud-native design, making use of cloud-born technologies under the hood such as auto-scaling infrastructure components, cloud storage, cloud analytics, containers, and serverless technology. This approach enables support for legacy and modern environments as your organization scales up and out with new architectures. No API security offering that deploys as a stand-alone, on-premises service can retain enough data necessary to inform baselines, drive analysis engines, and identify anomalous API traffic that indicates potential attack, privacy impact, or other type of incident.

Key architecture attributes that an API security offering should exhibit include:

1. **Environment agnostic** - the API security offering should support modern and legacy technologies regardless of where they are hosted. The offering should support integration into newer infrastructure like containers, Kubernetes, and service mesh, but it should also work in bare metal, virtual machine, and data center deployments. The offering should also be able to integrate with network elements like load balancers, API gateways, WAFs, and more. These integrations support distributed collection of API traffic across your enterprise architecture that invariably exists both on-premises and in cloud. The API security offering also needs to work in environments where traffic is encrypted since TLS is pushed readily as security best practice but has an unintended side effect of reduced security visibility.

2. **Independence from client-side code, server agents, and additional proxies** - the API security offering should not require additional server agents or network proxies. Most organizations already have too many proxies, which complicates troubleshooting and availability. Agents also have a bad reputation for creating deployment headaches and reducing performance. These issues become more apparent as organizations move towards new design patterns like microservices architecture where low latency and high throughput are essential. The offering should also avoid the use of client-side code or controls to stop attacks. These come in many forms including CAPTCHAs, SDKs, libraries, or JavaScript in the traffic stream. These approaches can create issues with front end performance or user experience, and they also don't work in direct API integration scenarios. Client-side controls are also defeatable by attackers.

3. **Cloud-based storage and analytics** - the API security offering should make use of cloud-based storage and data analytics, oftentimes referred to more

simply as Big Data. This approach is the only way to retain enough data necessary to inform baselines of API behaviors and consumption patterns, drive analysis engines, and identify anomalous events that are indicators of potential data loss, privacy impact, or other incidents. The large scale data collection and analysis at multiple points of your architecture and throughout the API lifecycle ultimately helps provide context for how your organization uniquely builds and integrates APIs as well as how those APIs are consumed.

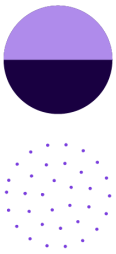
4. **AI/ML based analysis** - the API security offering should use AI/ML to analyze all the data and telemetry that are collected, produce meaningful signals, and inform security capabilities in the offering. Machine-assisted approaches are essential for powering detection and enforcement capabilities of any API security offering, such as determining where best to mitigate an API issue or what control is most appropriate. It may be more appropriate to revoke an attacker's authenticated session than to enforce a blanket rate limit that could inadvertently impact legitimate users. Machine-assisted analysis also helps reduce high false positive rates that are far too common with traditional security tooling.

API and sensitive data discovery

API and sensitive API and sensitive data discovery features should:

- Work across hosting models, environment types, and enterprise architectures
- Support legacy and modern API protocols
- Include API metadata beyond basic IP address and host information
- Identify API communications where sensitive data types are accepted or transmitted

API discovery and cataloging features support an organization's ability to identify all of its APIs so that the organization can in turn monitor and protect them. The catalog that exists in the organization's API management platform, if it uses one, is likely incomplete. Configuration management and asset management databases are either too stale or too far removed from API context.



Some organizations attempt to repurpose logging, vulnerable assessment, or network traffic analysis tooling, but the importance of purpose-built tooling to automatically collect and organize API metadata can't be overstated. While most existing scanning tools focus on IP address and host information, effective API discovery and cataloging must also include all appropriate API metadata such as API endpoints, API functions, path structures, message body structures, and more.

Like all security domains, the old adage applies that you can only secure (or protect) what you know about. Discovery and cataloging also provides benefits beyond security and includes governance, compliance, and privacy.

Core capabilities you should look for in an API security offering for discovery and cataloging include:

1. **Continuous, automated API discovery** - any API security offering should be able to automatically collect data and metadata about APIs across environment types. This discovery capability should be based on actual traffic and not just schema definitions since there is often deviation between documented design and the actual API deployment within organizations. The security offering should also collect this data, correlate it and organize the catalog continuously since the API landscape constantly shifts as a result of modern application design, system engineering, and IT practices.

2. **Identification of shadow or undocumented APIs** - any API security offering should be able to identify shadow APIs, also referred as unknown or undocumented APIs, that have flown under the radar of operations and security teams. This includes both shadow API endpoints as well as shadow API functions and parameters. An organization's API inventory is more than the APIs that it mediates with API gateways or publishes within API management offerings. There is a large ecosystem of APIs that are inherited as part of acquisition, integration, and cloud-native design. API development may also be outsourced or offshored, worsening the documentation problem. Build and delivery of APIs into production may also be less formalized. All of these factors feed into the problem of shadow APIs and a large, unknown API attack surface for organizations.

3. **Identification of zombie or out-of-date APIs** - any API security offering should be able to identify zombie APIs, also referred to as outdated or deprecated APIs. The life cycle of APIs inevitably results in multiple versions. However, old versions and old code of APIs often linger when building or operating APIs at scale. Organizations will often practice a type of version control with API keys, "restricting access" to old API versions by cycling out old API keys. In some cases, version control may even be as basic as adjusting request parameters which are controllable by the API caller. Zombie endpoints can contain buggy or vulnerable code, may expose excessive data or functionality, may no longer be monitored, and may lack production mitigations from other infrastructure. As a result, zombie API endpoints pose a significant risk to organizations and are often sought out by attackers.

4. **Identification of sensitive data** - as part of the discovery functionality, any API security offering should be able to identify sensitive data types in API parameters and payloads as well as tag API endpoints appropriately. There is a large range of personally identifiable information (PII) and also other data types that are subject to regulation. This includes protected health information (PHI) as defined by the Health Insurance Portability and Accountability Act (HIPAA) and cardholder data as defined by the Payment Card Industry Data Security Standard (PCI DSS). There are clear regulatory impacts to organizations when they inadvertently expose such sensitive data in API traffic. Newer privacy regulations such as General Data Protection Regulation (GDPR) and California Consumer Privacy Act (CCPA) are more expansive in their definitions of what constitutes private data and how organizations should protect such data. Failing to protect sensitive data can result in fees from regulatory bodies, severe brand damage, or lost customers.

5. **Identification of third-party API consumption** - any API security offering should be able to identify third-party API consumption and distinguish it from first-party API traffic. Applications are pieced together by integrating services and data of multiple APIs, not all of which are hosted by the owning organization. Third-party API consumption includes cloud services such as Google, Facebook, Slack, Twitter, and many others. It is also a byproduct of digital supply chains and partner ecosystems. Third-party API consumption is different from the more well-understood pattern of employee consumption of cloud SaaS services where CASB is typically the security control of choice. Modern design patterns result in a spiderweb of API interconnectedness, and it is common to see direct API communication and machine identities consuming data and functionality of third-party APIs. As a result, identification, behavior analysis and anomaly detection require different techniques than what is provided by CASB offerings.

API attack detection

API attack detect features should:

- Identify attacks against APIs quickly and early in attacker reconnaissance phases
- Support API schema analysis for design-time detection but also work independently of it
- Correlate anomalous API behaviors and attacker campaigns
- Work independently of traditional threat intelligence and malicious IP address feeds

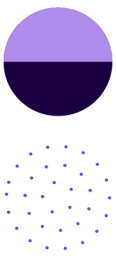
API attack detection is the ability of an offering to identify API attacks quickly and early. Traditional approaches with WAFs and API gateways fail to keep up due to their limited view of the API ecosystem. WAFs and API gateways focus on transactions in isolation and cannot see a complete API sequence to provide full context. API gateways, including those that exist as components of API management and iPaaS, are also primarily API mediators and access control enforcers that may already be overloaded in the enterprise architecture. Some organizations may attempt to repurpose IPS and NGFW for API security, but these are even less suited for the task of API attack detection since they sacrifice any application-layer or API focus for broad, multi-protocol attack detection.

Schema-dependent API security offerings fail at detecting certain types of API attacks, such as the ever prominent BOLA. There are inherent limitations with restricting analysis to only API documentation and schema definitions at the expense of also examining traffic in runtime. It's also possible to publish an API without any schema, schema definitions need not be granular such as with integer fields, and most organizations experience API drift as they ramp up with building, integrating, publishing, and operating APIs.

Core capabilities you should look for in an API security offering for API attack detection include:

1. **Attacker correlation** - the API security offering should be able to aggregate and correlate API traffic and associate it to attacker campaigns where applicable. Traffic collection seems like a simple task until you consider the volume of data that must be collected at a high frequency rate and continuously analyzed. Gathering such telemetry can only be supported with cloud scale data storage and log streaming (i.e., Big Data). Analyzing all that data at scale to unearth useful signals can only be accomplished with machine assistance (i.e., ML and AI). An API security offering should be able to correlate attack behavior per source IP address, per user ID, and per session ID and make that information readily available to API and security teams within the organization.

2. **Static metadata independence** - the API security offering should not be dependent on static data sources such as threat intelligence (TI) feeds or API schema definitions. TI feeds largely contain IP address and host information for known malicious entities on the internet. Organizations using traditional security controls often make use of these feeds as input into IP address allow and deny lists. Unfortunately, while they may be helpful for mitigating certain DDoS type attack patterns from botnets, they contain very little useful context for application security, let alone API security. In even basic automated attacks, attackers cycle through IP addresses or spin up compute within trusted cloud service providers to evade these basic



types of traffic management and network access controls. API schema definitions are often incomplete if not entirely absent in many organizations. If they exist, security teams may not have visibility into them depending on where the organization is at with DevOps maturity and siloed IT teams. TI feeds and schema definitions contain very little information that is useful for understanding the unique business logic of each organization.

3. **Behavior analysis and anomaly detection** - any API security offering be able to programmatically parse API business logic and behaviors in order to assess impacts to an organization's API security posture. While build time or pre-deployment checks can be valuable to overall security, a deeper contextual understanding of APIs can only be accomplished through runtime analysis. An API security offering should exhibit user and entity behavior analytics (UEBA) traits. UEBA-like capabilities are commonly found in some CASBs or SIEMs. In the case of API security, such anomaly detection capabilities must be tailored to different use cases and for APIs specifically. The anomaly detection should be able to detect a wide range of API abuses and automated attacks where API consumption patterns deviate from the baseline, or "normal."

4. **Early attacker identification** - the API security offering should not only be able to quickly and continuously detect API attacks, it should also be able to do it early. Typically, attackers go through an early reconnaissance phase as they passively and stealthily probe their target. This may be through throttled and distributed port scans to find exposed services and APIs. Attackers also commonly reverse engineer client-side application code, usually browser-based JavaScript or mobile binaries, to understand how backend APIs function and how to communicate with them. Such passive analysis techniques evade most detections since they typically appear as legitimate traffic. The API security offering should be able to detect subtle variations in normal consumption patterns that result from automation scripts and reverse engineering tools employed by attackers.

API attack prevention and blocking

API attack prevention and blocking features should:

- Stop API attacks before attackers can exfiltrate data or do damage
- Pair with attack detection capabilities so that API security is not passive or reactive
- Prevent security issues and misconfigurations from making their way to production
- Integrate with existing proxies in the enterprise for enforcement

Application security tooling such as static or dynamic analyzers and other forms of pre-prod scans help identify only a limited subset of API security issues. These tactics cannot stop attackers from exploiting those vulnerabilities. Runtime protection for APIs is necessary to prevent and stop API attacks. Unfortunately, some API security offerings provide detection only, with no ability to prevent or stop attacks, which creates market confusion.

Core capabilities you should look for in an API security offering for attack prevention include:

1. **Stop attacks that exploit OWASP API Security Top 10 (2019) issues** - any API security offering should be able to stop attackers that target the exploitable issues defined in the OWASP API Security Top 10. Most critical are BOLA attacks, previously referred to as insecure direct object reference (IDOR) attacks. An offering should be able to detect attacks that target authentication as well as the other type of authorization attack, broken function level authorization. An offering should also be able to detect excessive data exposure, lack of resource or rate limiting, security misconfigurations, injection flaws, and mass assignment flaws. While top 10 items like improper assets management and insufficient logging & monitoring aren't directly exploitable, attackers do target APIs that aren't adequately inventoried or monitored by organizations.

2. **Block malicious requests while learning and profiling** - any API security offering should be able to block or mitigate API attacks while it is profiling API traffic and learning the organization's unique business logic powered by APIs. There are a number of API attacks that can be detected and stopped regardless of how an organization designs and codes its APIs. This at a minimum includes injection-style attacks that follow well-defined patterns like XSS, SQL injection, XML injection, and JSON injection. It also includes excessive API consumption where API callers are consuming APIs or data at high volumes. Traditional rate limiting and message filtering mechanisms that you would expect to find in API gateways or WAFs are often too static, too operationally complex, or not well-maintained by the vendor.

3. **Stop credential stuffing and brute forcing attacks** - any API security offering should be able to stop these automated attacks that seek to achieve account takeover (ATO). ATO is a risk for all industries and any organization that exposes an API where authentication and authorization are required. Organizations will often invest heavily in strong access controls only to find attackers are finding ways in by pilfering credentials. Brute force attacks are the more well understood pattern, where attackers try many sequences of usernames and passwords in an attempt to find

working credentials. Credential stuffing is a newer pattern, where attackers harvest credentials from prior breaches and repurpose them in new automated attacks against other organizations. Credential stuffing is often successful because username and password re-use is commonplace. Even in cases where additional authentication factors are used, such as a 2FA authenticator or SMS challenge, attackers will combine credential stuffing with brute forcing to overcome these stronger access control approaches.

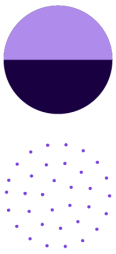
4. **Stop application-layer denial of service (DoS) attacks** - any API security offering should be able to stop application-layer DoS. DoS and distributed DoS (DDoS) are often viewed from the lens of excessive traffic or request rates, or volumetric attacks. The volumetric attack pattern is only one form of DoS and DDoS. It is also readily addressed by large scale CDNs and cloud service providers that can absorb and mitigate such high volumes of traffic. The more nefarious and stealthy form of DoS is application-layer DoS, or layer 7 DoS. Application-layer DoS is more difficult to detect and stop because of application and API uniqueness. Many offerings and service providers will mitigate layer 3 and 4 DoS and DDoS readily but leave an organization exposed for layer 7 DoS. A given security tool must analyze the organization's APIs and API traffic to effectively prevent such attacks.

API-centric incident response

API-centric incident response features should:

- Integrate with existing work streams and dev and SecOps tooling
- Provide custom views and workflows for many IT personas, not just security
- Address many incident types including privacy impacts and availability problems
- Support integration with the organization's ITSM, SIEM, and SOAR implementations

Attacks are inevitable and organizations must deal with threat actors on multiple fronts. External or public APIs are prime targets, but so too are internal or private APIs where security controls may be more lax in favor of protection provided by traditional perimeter controls. While API protection is key to defending your APIs in runtime, the organization's ability to respond in the event of an attack is just as critical. Not all API-related risks are attack-oriented either where concern may be data exfiltration or scraping by an attacker (i.e., a data breach). Incidents encompasses many unforeseen events including unintentional data exposure, privacy impacts, and availability issues.



Any API security offering you consider should support basic alerting methods. Email and SMS are tried and true notification mechanisms that may appear in procurement checklists and RFPs. More importantly though, an API security offering should provide integrations with the organization's pre-existing IT systems and workflows. Organizations need modern notification and response techniques to shorten mean-time-to-detect an incident and mean-time-to-repair. Look for integrations and automation that support your already overwhelmed SOCs or augment what your MSSP is able to provide. Data feeds into the organization's SIEM are a given, though it should be done intelligently to provide useful signals. An API security offering should also support newer SecOps capabilities like SOAR.

Core capabilities you should look for in an API security offering for API attack response include:

1. **Intelligent ITSM, SIEM and SOAR integration** - the API security offering should provide integration into these commonly found IT and SecOps systems. Integration should not be limited to a basic "log feed" or "data dump.". Rather, the API security offering should intelligently prioritize events, provide actionable security alerts, and support the work streams of a modern SOC and IT workforce. The offering should be able to trigger workflow within external SIEM and SOAR offerings, such as Splunk and Demisto. It should also be able to trigger workflow within external ITSM for ticketing, such as ServiceNow and ITSM Atlassian Jira Service Desk.

2. **Customizable response actions** - rather than depending on a native integration, the API security offering should provide APIs or webhooks to integrate with a wider range of external IT system systems. Integration should support customizable response actions, and complex, multi-party workflows.

3. **Tailored for multiple IT personas** - the API security offering should provide a role-based access control model (and that can be integrated with external IAM) that allows for varied levels of view and control. It should be possible to delegate functions to different users or groups of users. As an example, the organization would likely want to expose any remediation insights to development teams, but only for those API endpoints they are responsible for. The UI and UX should be customizable for various roles so that only desirable data and functionality is presented.

API vulnerability identification and remediation insights

API vulnerability identification and remediation insights features should:

- Check for a wide spectrum of API security issues, misconfigurations, and vulnerabilities
- Expand detection beyond known vulnerabilities
- Support checks that can be triggered in development, build, and runtime phases
- Run continuously and automatically for the entire lifecycle of APIs

Traditional vulnerability scanning is focused on finding known vulnerabilities in published software or hardware. Typically, this results in flaws or misconfigurations that map to CVE IDs. Oftentimes, there are simply too many of these scan results to act on in a timely fashion, which has plagued vulnerability management programs for decades and fueled a lot of the desire to "shift-left." Some CVE IDs may not be fixable since it involves third-party code, and some flaws may not even be exploitable depending if the relevant vulnerable code is reachable within your organization's unique application design and serving architecture.

In the world of custom software development and custom API work, CVEs lose relevance beyond third-party dependency checking. Organizations frequently source software from commercial vendors and open-source projects, and that componentry may contain latent vulnerabilities. However, there is also a wide spectrum of design flaws, software weaknesses, business logic flaws, and more that do not map neatly to CVE IDs. These are unknown vulnerabilities in unpublished software, and responsibility lies with the organization that created the code or integrations to fix a given issue. A security fix may not always be code-level, since it may not be technically possible to do so, it may not be feasible to produce a code fix in a timely manner, or it is more practical to mitigate through other infrastructure components.

Core capabilities you should look for in an API security offering for API vulnerability identification and remediation include:

1. **API vulnerability and weakness identification** - the API security offering should use a combination of techniques to assess the security of the APIs that the organization hosts, integrates and consumes (i.e., third-party API consumption). The offering should passively analyze API traffic that flows through numerous points of enterprise architecture on and off-premises, and it should analyze API schema definitions when available to identify

areas of API weakness that should be remediated by development (i..e, code a fix) or operations teams (i.e., mitigated by infrastructure). Any API security offering should also be able to analyze traffic in pre-production environments to limit the risk of incidents in production environments or data exposure for production users.

2. **Remediation guidance tailored to developer and operations perspectives** - the API security offering should provide remediation guidance focused on code-level fixes for development perspectives as well as infrastructure configurations for operations perspectives. Issues should be mapped to the OWASP API Security Top 10 where appropriate, but technical details should not be limited to just security context or written for security audiences.

3. **Integration with external defect tracking** - the API security offering should provide basic remediation tracking for identified issues, but more importantly, it should integrate with external defect tracking systems in order to support pre-existing security and development workflows for remediation. Defect tracking may be handled in external DevOps solutions, such as Azure DevOps or Atlassian Jira. Defects may also be tracked in external ITSM or vulnerability management (VM) platforms depending on the organization's security program.

4. **Code repository, build system, and delivery system integration** - the API security offering should provide a mechanism to integrate with development, build, and release systems. This capability may come through version control system integration and git-based code repositories to statically analyze API code or schema definitions. It may also be through build scripts or CI/CD integration to dynamically analyze APIs in runtime in pre-production or production environments. It should also be possible to pass or fail builds based on what the API security offering finds. Ideally, integration is provided via API, webhook, or native integration with the relevant build system, but command line invocation may be an alternative.

Conclusion

Salt Security is providing this research to industry to improve awareness of what it takes to adequately secure APIs and provide guidance on what to look for in a given API security offering. We want to enable decision makers within organizations to filter vendor claims and map to necessary security functionality to protect their business. Your organization may be vulnerable if it is relying on some of the traditional approaches like application security testing, traffic management or API threat protection from gateways and WAFs. API context matters greatly. Traditional approaches might check off some API security features but not all. Application

security testing tools can't surface business logic flaws. WAFs and API gateways focus on transactions in isolation rather than analyzing complete API sequences to detect business logic abuse.

Underlying design and architecture of an API security offering matters. Any offering you are considering should be architected as a platform and address the full lifecycle of APIs to secure them appropriately. API security cannot be packaged as a stand-alone tool or scanner due to the stages at which security and privacy issues manifest themselves. A given problem may never result from the code of an API. Rather, it may be your configuration or implementation of that API that results in a security or privacy risk. It may also be an unintended API design as part of a complete system that allows for business logic abuse by attackers.

The information in this evaluation guide is derived from an extensive RFP toolkit we have built and provide to potential customers as part of sales engagements. You can contact us at any time for further information or a demo of the Salt Security API Protection Platform.

Salt Security Platform

Only Salt Security delivers the context you need to protect your APIs across build, deploy, and runtime phases. We combine complete coverage and an ML/AI-driven big data engine to provide that context to show you all your APIs, stop attackers during the early stages of an attempted attack, and share insights to improve API security posture.

The Salt approach

Salt deploys in minutes and automatically discovers all your APIs and where they expose data, pinpoints and blocks attackers, and provides remediation insights for dev teams.

Our advantages derive from our C-3A Context-based API Analysis Architecture – with coverage across all your app environments and our big data engine powered by our time-tested ML and AI algorithms.

Complete coverage

We collect all your API traffic – across load balancers, API gateways, WAFs, Kubernetes clusters, cloud VPCs, and app servers - to dynamically provide a full inventory. We deploy with no app or network changes and require no configuration or tuning.

AI-powered big data engine

Every one of your APIs is unique. Salt applies ML and AI in our big data engine to baseline your APIs and isolate anomalous behavior, differentiating between changes to APIs and malicious activity. By applying the context we learn, we can avoid false positives.

Context-based analysis

Salt combines our complete coverage and big data engine to discover all your APIs, see the sensitive data they expose, find and stop attackers, and capture insights for development teams to improve your API security posture.

Additional Reading

Securing APIs – It's Different Than Securing Apps

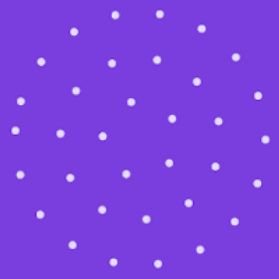
3 Reasons You Might Be Failing at API Security

How Shift-Left Extremism is Harming your API Security Strategy

Stopping API Attacks: Columbo, Correlation, and Context

Is OAS Enough For API Security?

Extending our Lead in API Security – Augmenting our "Shift Left" Features



SALT

Securing your Innovation.