National Defense ISAC

API Services Security

April 4, 2022

Authors:
Raul Barreras
Paul Keim
Isaac Lenox
Richard Macantonio
Waldemar Pabon
Joe Vega
Andrew Zuehlke

Reviewers:
Will Jimenez
Renee Stegman

About the Authors

Dr. Waldemar Pabon, ND-ISAC Lead & Senior Contributor

Dr. Waldemar Pabon is a Cyber Security Architect with over 27 years of experience in software engineering. Dr. Pabon leads the Application Security Working Group, Software Security Automation and the COTS Software Assessments Subgroups at the ND-ISAC. Under his leadership, the Software Security Automation Working Group has published four white papers. The papers provide ND-ISAC members and the industry with a roadmap on how to adopt application security best practices while leveraging automation as a catalyst to achieve efficiencies. Dr. Pabon has a Doctor of Science in Cybersecurity degree from Capitol Technology University.

Raul Barreras, ND-ISAC Contributor

Raul is an Information Security Professional with more than 20 years of experience. His multiple roles throughout his career include information security officer, systems administrator, developer, teacher, and occasional pen tester.

In recent years, Raul has had the opportunity to use the accumulated experience in a new role: application security. This role has allowed him to discover how much he enjoys being a developer advocate and how fun and useful it is to build a community of security champions, while improving the quality of the organization's software. This is the second opportunity Raul had to contribute to application security papers at the ND-ISAC.

Paul Keim, ND-ISAC Senior Contributor

Paul Keim is a Senior Security Architect. With ten years of experience in the information security field, with half of that leading the Application Security team, Paul's focus is constantly on how to automate his processes to reduce the workload for his team. A member of ND-ISAC for three years, Paul started in the Application Security subgroup looking to learn more about what other companies are doing to automate their workloads, and now looks to provide his expertise on what's worked and hasn't worked for him over the years.

Isaac Lenox, ND-ISAC Contributor

Isaac Lenox works as a Security Engineering Group Director supporting the Global Security Operations Center and other departments with full stack application development and system engineering. He has over ten years of experience spanning disciplines such as penetration testing, software development, incident response, exploit development, automation, threat intelligence, and DevOps. His focus now is on compliance automation, secure and efficient development operations, and designing advanced network defense capabilities. Isaac has been a member of ND-ISAC for five years, attending Live! events and participating in threat intelligence automation. He joined the Application Security Working Group looking to contribute his expertise and expand his perspective.

Richard Marcantonio, ND-ISAC Contributor

Rick Marcantonio is a Cyber Security Engineer supporting a myriad of technologies. Rick is a US Navy veteran and holds degrees from Tarleton State, Embry-Riddle, and an MBA from Dallas Baptist University. For Rick, this white paper represents one of his many contributions to the ND-ISAC community via working groups and peer collaboration.

Joe Vega, ND-ISAC Senior Contributor

For the last four years, Joe Vega, has had the role of Cyber Security Engineer. Receiving Bachelor's degree in both Cybersecurity (Information Assurance) and Information Systems from the University of Texas at San Antonio (UTSA), while also obtaining his CISSP. Joe has over ten years of experience within the cybersecurity industry, with five years specific to the Application Security field. In his current role, Joe is focusing on Application Security and DevOps workflows. Joe has been a member of ND-ISAC for two years, contributing to two papers in the AppSec space. Joe continues to actively participate in the ND-ISAC community offering his expertise to his peers in the field.

Andrew Zuehlke, ND-ISAC Senior Contributor

Andrew Zuehlke graduated from Appalachian State University in 2017 with a Bachelor of Science Degree in Computer Science and Computational Mathematics. Andrew has served as the lead administrator of SIEM and EDR solutions. In early 2020, Andrew moved to his current role as Cybersecurity Liaison, primarily supporting Research & Development and Information

Technology. Since joining ND-ISAC in December 2020, Andrew has become a member of the Application Security working group as well as the Cloud Security and Architecture Working Group. As his first major contribution, Andrew coauthored the Application Security Working Group's third white paper, Remediation Workflow Automation. Andrew continues to contribute to the ND-ISAC community via working groups and peer collaboration.

Executive Summary

The use of application program interface (API) Services has revolutionized the way applications are developed and used. API Services provide a separation of the user interface (UI) and the backend concerns as they support the business logic without the need for constraints governing the UI. This separation has benefited automation efforts since critical features in applications can be consumed on demand enabling efficiency and agility. Furthermore, because of standardization and uniformity in their implementation, API Services have empowered new architectures, such as microservices, which will simplify the solution architecture and enable the provisioning of new features and fixes at a faster pace.

From a security perspective, API Services are prone to threat vectors just like any other type of application. Even though the user interface threat vectors are limited, because of their focus in the interaction with backend processes, this inherent behavior could enable API Services to compromise the technology infrastructure. This reality stresses the importance of implementing security controls in the Software Development Lifecycle (SDLC) to ensure these potential threats are identified and remediated before services are rolled out to a production environment. This explains the reason why the OWASP (Open Web Application Security Project) organization has released a Top 10 list of threat vectors specifically for API Services[1].

[1] https://owasp.org/www-project-top-ten/

Having security as the primary driver, this paper provides guidance on how to establish a governance process enabling the API Services to share a common security posture and controls. The paper discusses how the use of an API Gateway can provide governance and standardization in the implementation and security of the API Services. The API Gateway is the centralized structure to ensure a sound and secure implementation of services throughout the organization. In order to define the strategies needed in the API Gateway to ensure a standard and secure use of services, it is imperative to understand the data these services will have access to as well as the business impact associated with its functionality. An API Gateway provides authentication, authorization, threat detection as well as threat protection capabilities for API Services.

If the organization wants to have better control over the code as well as those services provided by third-party entities, it is critical to understand the difference in terms of internally developed as well as external API Services. This difference requires the implementation of additional measures to ensure the validation as well as consumption of those external API Services are compliant with sound security standards. Even when remediation of potential threats found in third-party services may be limited, organizations are responsible for minimizing the risk associated with the potential threats that may exist in these services. Here, security controls such as the Dynamic Application Security Testing (DAST) will provide great visibility into existing vulnerabilities and the proactive protections that may be needed to minimize the risk to the organization. Managing the access to these third-party services

through the API Gateway will enable the organization to have the same level of control as the one provided for internally developed services.

Table of Contents

Introduction

Objective

Technology innovation has provided organizations with new architectures to simplify the provisioning of features to the end-users in different platforms. Whether a mobile application or a container, new architectures based on API Services are allowing users and systems to be able to communicate with the infrastructure with a high degree of flexibility and agility. API Services represent a narrow scope of an implementation. To understand what API Services are, it becomes important to understand how the industry has transitioned from legacy applications.

In legacy applications, the deployment of new features was encompassed by a single deployment structure. All the features were deployed at once as part of the single unit of deployment. If a feature needed to be changed or fixed, the deployment would require teams to validate all the features because as part of the deployment process, all features are encapsulated within the same set of files. This created overhead in scenarios where only one feature or fix is introduced. This becomes especially important in agile environments where single features are pushed to production in a recurring fashion.

With API Services, development teams can break down into single units the implementation of features. If a new feature is needed or a feature requires a bug fix, implementation teams can test and deploy that single unit of functionality without impacting any of the existing features in

the production environment. This type of scenario provides efficiency and flexibility, which is not present in monolithic deployments. The use of API Services provides support to new architectures aimed at providing more agility and supporting DevOps implementations.

API Services are prone to security vulnerabilities, similar to legacy monolithic applications, because they also represent software components. Since the API Services provide backend support in implementations, much of the security risks associated with front-end vulnerabilities are less of a concern. Leveraging API Services minimizes the impact of security vulnerabilities at the user interface level but does not eliminate all risk. The services are working with backend components; thus, attackers could leverage threat vectors which could reach the underlying platform easier. This is one of the reasons why the OWASP organization published a new set of threat vectors exclusively for API Services. A quick review of these threats will reveal how vulnerabilities are impacting implementation at the backend (OWASP, n.d.).

Dealing with these threats and securing the implementation requires a combination of security controls as well as components providing governance over the API Services implementation. The main objective of this paper is to precisely provide an overview of how to secure API Services, which threat vectors are impacting their implementation, and how an API Gateway could be used as a bridge to tie the API Service security strategy altogether. This paper provides a transition from the required security controls in the Software Development Lifecycle to the protection of API Services.

Audience

The audience for this white paper includes security engineers, software engineers and software architects responsible for the design, implementation, and protection of API Services in the technology infrastructure of any type of organization, regardless of size or industry.

Structure of the paper

This paper introduces the security controls needed to protect the code behind the API Services. An explanation of the common threat vectors is provided to bring awareness to the reader of the security risks impacting API Services from a high-level perspective. The paper discusses how the use of an API Gateway can provide governance and standardization in the implementation and security of the API Services. The major features of the API Gateway are explained as well as the special considerations regarding the use of API Keys and the protection of external API Services.

API Services Security

Security Controls in the Software Development Lifecycle (SDLC)

API Services provide multiple flexible venues for organizations. The ability to decompose the features of implementations in unique software artifacts provides a support line for newer technology architectures. To create API Services, software development teams need to follow similar strategies used in other types of applications. Code will be stored in a source code manager, 3rd party components, or libraries that will be used to accelerate the time to market, or other business needs. Based on the software, API Services are also prone to security vulnerabilities and threats because of the nature of the implementation

This reality provides a justification for understanding how important it is to secure the code being implemented by development teams whenever they are creating API Services. As detailed in ND-ISAC's previous papers, "Software Security Automation: A Roadmap toward Efficiency and Security" and "Software Security Automation: Security Controls Evaluation Criteria", implementing security controls in the SDLC is a requirement when securing the software. [2]The importance of these security controls resides in the ability to minimize the risk of software weaponization by supporting an early detection approach.

Each of these security controls will provide all the detection and the associated information needed by development teams to address any security threat or vulnerability before the

[2]https://ndisac.org/wp-content/uploads/ndisac-security-automation-white-paper.pdf
https://ndisac.org/wp-content/uploads/2020/10/ND-ISAC-Software-Security-Automation-Security-Controls-Evaluation-Criteria-Final_Oct2020.pdf

application is deployed to a production environment and potentially exploited by an attacker. Regardless of the type of API Services created, these security controls are designed to improve the security stance of the software at the different stages of the SDLC process. Table 1 provides a summary of these security controls and their importance in the SDLC.

Table 1: Application Security Controls in the SDLC

Technology	Description	Type of Vulnerabilities Detected
Static Application Security Testing (SAST)	Conducts white box testing, performing analysis of source code for security vulnerabilities early in the software development process as part of the Integrated Development Environment (IDE), the commit or build process in a Development Operations (DevOps) methodology.	Vulnerabilities in the static source code software
Software Composition Analysis (SCA)	Provides detection capabilities for security vulnerabilities in third-party components.	Vulnerabilities in any third-party libraries imported
Dynamic Application Security Testing (DAST)	Conducts black box testing to detect vulnerabilities associated with the application behavior by evaluating content from user/attacker perspective while the application is running.	Vulnerabilities and runtime problems in a running application
Runtime Application Self Protection (RASP)	Provides detection and protection capabilities during runtime through instrumentation by monitoring an application's behavior and context of the behavior.	Vulnerabilities that may not have been identified in the previous scanning solutions
Interactive Application Security Testing (IAST)	Analyzes code by scanning for security vulnerabilities while the application is being tested (automated or manual test). This security control runs outside of the continuous integration continuous delivery (CI/CD) pipeline.	A wide variety of vulnerabilities from anywhere in the development process.

Identify API Discovery

Users cannot protect what is unknown or undiscovered. Having an appropriate inventory is the first step in being able to protect assets properly. Developers are highly recommended to register APIs in an API registry, like The Registry API (Github, 2022). The use of machine-readable descriptions is of paramount importance to make maintenance easier.

Inventories have the bad habit of being incomplete or out of date and sometimes developers could forget to register their APIs, leading to documentation drift, so we need to have an API discovery process added to the inception process. Such a process must have the following features:

- Continuous. An inventory will become obsolete at the time it is ready to be used. It needs to be updated it regularly.
- Automated. The number of APIs in use is constantly increasing, and not always in an organized way.
- Shadow APIs are identified. Undocumented APIs are a huge risk and increase the organization's attack surface.
- Outdated APIs are identified. Outdated APIs may contain bugs or lack security controls.
- Sensitive data is labeled and identified. Failures protecting sensitive data can bring economic and legal consequences and affect the brand image. Knowing where

sensitive information is handled allows developers to correctly place the appropriate controls to protect it.

- 3rd party API. They are part of the attack surface.

Inventory must contain not only the production APIs, but also the development and testing APIs. Each tool that collects API data (API Data Collectors) must process the gathered data and generate appropriate API metadata, including information about API endpoints, API functions, parameters, version number, and data schemas used. Some fields to be considered:

- API Name
- API Description
- API Protocol. It could be SOAP, REST, GraphQL, RPC, etc.
- Host and Port used
- API functions
- Path structures
- Message body structures
- API Dependencies
- Data sensitivity
- Defect tracking system associated
- Clients

API Data Collectors

Collecting data about APIs can be done at various stages of the software development lifecycle. Many tools, like SCA (Software Composition Analysis), Security Information and Event Management (SIEM), vulnerability scanners, and network traffic analyzers, can be repurposed for API discovery.

- Source code analysis. Can validate data schemas, search for vulnerabilities, and help with the documentation
- Reverse and forward proxies. Can detect shadow APIs and/or update current APIs.
- API Gateways. Can validate data schemas and identify sensitive data.
- Network traffic analyzers. Can detect shadow APIs, update current APIs, and data sensitivity policies violations.
- Web servers' logs. Can detect shadow APIs, update current APIs, and vulnerabilities based on the response code and payload used by attackers.

Categorization for the APIs

Establishing a process for categorizing the APIs in the environment will assist developers in both planning controls and understanding assets. Below is a non-exhaustive list of categories (Table 2) developers can us to build an overall categorization framework. The table shares

some of the more common criteria for categorizing APIs, other categorization mechanisms are dependent upon the environment.

User Base

This category depends on the question, "Who should be accessing these APIs?". Oftentimes, audiences will not be able to be strictly defined, so the recommendation is to use the largest target group for the customers. An example table (insert table number) of user bases is provided below:

Table 2: **User Base Categories**

User Base	Example Scenario
Business to Business ("B2B")	Information sharing between partner organizations.
Business to Customer ("B2C")	Customer-oriented single page web application using API driven functionality
Private	Internal cross-team data or business flows

Accessibility

This category asks, "Where should this API be accessed from?" This is usually driven by the specific user base of the APIs and is often practically informed by the desire to manage operational overhead of IP Allow Lists and other connectivity-limiting controls. An example list of connectivity types is provided in Table 3 below:

Table 3: **Connectivity Type Categories**

Connectivity Type	Example Scenario
Public	API that may need to be accessed from any network or location
Protected	API available to partner organizations only
Private	Internal cross-team data or business flows

Allowed Actions

This category asks, "What can the API do?" An endpoint that allows the updating or deletion of data can be more operationally sensitive than an endpoint that only allows certain data to be read. Oftentimes, these will be able to be tied to specific HTTP methods as well, but that will largely vary depending on development team sensibilities. Table 4 below explains standard HTTP actions/verbs/descriptions.

Table 4: **Allowed Actions in API Services**

Actions	Standard Related HTTP Verbs	Description
Read	GET POST	Ability to read / retrieve data
Update	POST PUT	Ability to update or create data
Delete	DELETE	Ability to delete data

Data Classification

This category asks, "What data is exposed by this API?" Specific data classifications and the related controls for this will vary significantly between organizations and industries. We recommend consulting with your legal and any data governance teams to help drive the

definition of these categories and what compliance obligations may impact controls. An example list of data classifications is listed in Table 5 below:

Table 5: **Data Classification in API Services**

Data Classification	Example Data Elements	Description
Legal Consequences	Social Security Numbers Personalized User Analytics	Data such as Personal Identifiable Information (PII) that, if lost, is required to be reported to a government agency or other business with potential legal (criminal or civil) consequences
Internal Data	Internal Hostnames Non-individually identifiable personal information	Data that has no specific legal ramifications around loss, other than potential loss of organizational reputation
Intellectual Property	Platform Architectures Protocol Information Marketing Analytics	Organizational intellectual property that could represent loss of operational advantage if lost or stolen
Marketing Material	Sales Pamphlets	Publicly available material that represents no loss to the company if freely available externally

Deployment

This category asks, “How is the API deployed?” This can cover a number of sub-categories depending on your organization; examples can include: API Gateway product used, data centers used, or Cloud environments the development organization use.

Other Categorizations

Some other categorizations that may be valuable depending on the organization's maturity or the planned set of controls include:

- Organizational Ownership – "Which organization in the company owns this API?"
- Development Methodology – "Is this developed by an agile or waterfall team?"
- Expected Client – "Is this expected to be interacted with by a web or mobile client?"

API Common Security Threats (OWASP API Top 10)

API Services are built with software and software is prone to security vulnerabilities. This explains why organizations such as OWASP has created a top 10 list of security threats affecting API Services (OWASP, n.d.). Some of the threat vectors listed in the OWASP top 10 are also common in other types of architectures (for example, we could find similar threats affecting web applications). Table 6 below provides a list of the OWASP API Top 10 threats.

Table 6: OWASP API Top 10 Threats

Threat Vector	Description
Broken Object Level Authorization	APIs tend to expose endpoints that handle object identifiers, creating a wide attack surface Level Access Control issue. Object level authorization checks should be considered in every function that accesses a data source using an input from the user.
Broken User Authentication	Authentication mechanisms are often implemented incorrectly, allowing attackers to compromise authentication tokens or to exploit implementation flaws to assume other user's identities temporarily or permanently. Compromising a system's ability to identify the client/user, compromises API security overall.
Excessive Data Exposure	Looking forward to generic implementations, developers tend to expose all object properties without considering their individual sensitivity, relying on clients to perform the data filtering before displaying it to the user. This vulnerability is very common and often results in unnecessary data being sent to the user while relying on client-side tools to "filter" the correct data for consumption.
Lack of Resources & Rate Limiting	Quite often, APIs do not impose any restrictions on the size or number of resources that can be requested by the client/user. Not only can this impact the API server performance, leading to Denial of Service (DoS), but also leaves the door open to authentication flaws such as brute force.
Broken Function Level Authorization	Complex access control policies with different hierarchies, groups, and roles, and an unclear separation between administrative and regular functions, tend to

	lead to authorization flaws. By exploiting these issues, attackers gain access to other users' resources and/or administrative functions.
Mass Assignment	Binding client provided data (e.g., JSON) to data models, without proper properties filtering based on an allow list, usually leads to Mass Assignment. Either guessing objects properties, exploring other API endpoints, reading the documentation, or providing additional object properties in request payloads, allows attackers to modify object properties they are not supposed to.
Security Misconfiguration	Security misconfiguration is commonly a result of unsecure default configurations, incomplete or ad-hoc configurations, open cloud storage, misconfigured HTTP headers, unnecessary HTTP methods, permissive Cross-Origin resource sharing (CORS), and verbose error messages containing sensitive information.
Injection	Injection flaws, such as SQL, NoSQL, Command Injection, etc., occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's malicious data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
Improper Assets Management	APIs tend to expose more endpoints than traditional web applications, making proper and updated documentation highly important. Proper hosts and deployed API versions inventory also play an important role to mitigate issues such as deprecated API versions and exposed debug endpoints.
Insufficient Logging & Monitoring	Insufficient logging and monitoring, coupled with missing or ineffective integration with

	incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems to tamper with, extract, or destroy data. Most breach studies demonstrate the time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.

API Gateway

What is an API Gateway?

An API Gateway is an API management tool that can provide governance and standardization in the implementation and security of API Services. Acting as a reverse proxy, an API Gateway accepts API calls, applies security features to each request, routes the requests to the correct service, and returns results back to the original requestor—all while maintaining a record of all activities. By acting as a central hub for all API calls, API Gateways take the responsibility of common actions or capabilities from individual developers and applies them across all API Services.

A common use case for API Gateways is authentication: when prompted to add secure authentication capabilities to their API Services, ten different developers may implement ten different authentication solutions. This not only results in duplicated effort, but developers may not integrate the same level of security. Instead of expecting developers to create

processes for authentication, an organization can utilize an API Gateway for authentication, ensuring consistent and secure authentication methods are applied across all API Services.

While API Gateways have the potential to offer numerous benefits to organizations with multiple API Services, it is important to note that it may not be the right solution for all situations. For example, because an API Gateway is, in itself, a tool, it requires resources to configure and maintain; therefore, it may not be effective to stand up an API Gateway for a single API Service.

Security and Health Configuration

As with all management solutions, some settings can be properly configured to secure APIs. Below are five foundational settings that are likely to be applicable in most API Gateway use cases, however it not an exhaustive list.

1. Authentication

Although not always required, it is generally best practice to enforce authentication instead of allowing anonymous access, especially when restricted or sensitive data is involved. Using an API Gateway, an organization can consistently apply authentication across all API calls routing through the gateway, eliminating duplicate effort of developers having to develop authentication processes.

2. Encryption

Encryption has become essential to protect all data while in flight. Regardless of whether authentication is utilized, all communications between the API Gateway and clients should be encrypted using Transport Layer Security (TLS). By enforcing encryption at the API Gateway, organizations can guarantee that all communications are sent over Hypertext Transfer Protocol Secure (HTTPS).

3. Rate Limiting

It is becoming more and more common for organizations to fall victim to denial-of-service (DoS) attacks, where malicious actors overload the infrastructure by initiating an unusually large number of requests. API Gateways can be configured to apply rate-limiting, or throttling, to API calls. API Gateways can also be used to enforce request and response size limiting. These limits help protect against unexpected spikes in requests as well as protect the organization against repeated calls from malicious actors or bots.

4. Logging and Statistics

Logging at the API Gateway level creates an audit trail for all requests being handled by the gateway. Not only can this help with forensic investigations, such as identifying who has accessed what data, but it can also be used to gather statistics on API usage.

5. Minimize Unintentional Data Exposure

Frequently enabled for debugging purposes during development, error codes often unintentionally expose information when left enabled in production. API Gateways can be configured to return appropriate error codes where necessary, minimizing the risk of a custom API returning excessive information through error codes.

API Threat Detection

As the central access point to every backend API Service, the API Gateway is a key location to detect threats. From this vantage point, every aspect of API traffic can be observed, tracked, and analyzed. This section details various areas of threat detection from an API Gateway and beyond.

Logging

The most critical function used to detect threats is effective and well-designed API Gateway logging. Without this, detection efforts will lack critical information needed to help support a more proactive risk reduction approach. Simply generating log events is not enough to detect threats; rather, effective, and informative log message generation must be carefully thought out and developed. Additionally, log messages must be centralized to facilitate correlation with other log sources. Suspected as well as imminent threats can be detected with effective log message generation, log centralization, and correlation of events..

Both the generation and contents of log messages need to be carefully evaluated. Beyond debugging logs, the API Gateway should log security events and user activity. Additionally, log messages need to describe at least what happened, what or who was involved (such as client identification, backend service destination, etc., and when the event occurred. In order to develop robust and effective logging within the API Gateway, the above logging techniques should be considered throughout the Software Development Life Cycle. If logs remain on the API Gateway system and are not transferred elsewhere, the threat detection capability is greatly diminished. Furthermore, this leaves the logs exposed to risks of corruption, destruction, and malicious tampering. To remedy these points, the API Gateway should be configured such that logs are transferred, if not streamed, to a log repository.

Base log messages as the only source are often not enough to consistently detect true positive threats. Once logs are generated for important events, contain required information, and are transferred to a central log repository, they can be correlated with each other and other log sources to generate alerts. These correlated alerts can be highly confident and greatly reduce false-positive threat detections.

Traffic Monitoring

Malicious activity often has many attributes that set it apart from legitimate activity. These differences range from the specific and betraying to the generalized and broad. Details of malicious activity such as the User-Agent of a scanner are very specific and can be detected

with a simple text match within the API Gateway or an Intrusion Detection System. However, broader characteristics of malicious activity such as the frequency of requests are more difficult to detect and are generally done with specialized monitoring software but can be built-in to the API Gateway.

Specific, known attributes of threats can be detected with signatures. These can be implemented in the API Gateway or an external security appliance. They can be created with simple text or string matching or advanced pattern matching. To create a signature, a sample of traffic from a known exploit or other threat needs to be analyzed. An attribute of the sample that is not common in legitimate traffic must be determined for the signature to be made to match. This kind of detection can be high confidence but usually cannot detect new or unknown exploits or other threats.

Often malicious traffic can be entirely unknown or have no specific differences from legitimate. In these cases, detection can be incredibly difficult. Defenders must use heuristic detection methods to increase detection likelihood in addition to what signature detections provide. This could involve detection of what is not known as legitimate traffic or otherwise uncommon traffic. If the API Services are only queried by corresponding client software (such as a JavaScript Web Application), the way this software interacts with the APIs can be used as a baseline. Any traffic that diverges from this baseline could be indicative of a threat.

Heuristic detection could also involve anticipating broad, common attributes of malicious traffic, such as rapid requests from a single source or odd parameters.

API calls are almost always transported with the HyperText Transfer Protocol (HTTP). This protocol has a variety of header fields with information to and from the browser, web server, API Gateway, and API Services. The HTTP header fields, and their values of both malicious and legitimate traffic can be analyzed to determine key differences to be used in monitoring. Every attribute from field order to listed fields to actual field values could be used to differentiate malicious from legitimate traffic.

Client Tracking

The behavior over time of clients can often distinguish legitimate from malicious clients. Monitoring, tracking, and analyzing all actions of each client can be done in several ways. Client details such as their IP address, a unique cookie, and cipher suites can be used to consistently identify clients in each action taken. Even simple statistics such as clients generating the most requests can be useful in detecting threats. Clients frequently querying uncommon or non-existent API endpoints can also be potential threats. While log repositories can be a great source of this information, they are not designed to analyze behavior over a long period.

Tracking the state of clients over time can prove difficult at scale, so a dedicated tool is usually needed. A specialized tool that stores and analyzes client actions over time can greatly improve threat detection capabilities. This kind of tool is usually programmatically integrated with the API Gateway, but a simple log feed can be sufficient as well. Finally, threat intelligence can be incorporated into monitoring to alert known malicious identities. There are intelligence feeds that can be ingested programmatically and utilized in the SIEM or directly in a feature-rich API Gateway. These feeds contain specific indicators, such as malicious IP addresses or JA3 client identifiers that can be alerted on if seen communicating with the API Gateway. This automatic data sharing can often detect threats before a compromise occurs.

API Threat Protection

API Services, just like any other piece of software, could be prone to security vulnerabilities and attacks. Therefore, it becomes important for the API Gateway to provide protections to the services included under their purview. Even though API Gateways could provide protections and immediate mitigation, it is important to understand that the real remediation for the security threats will come from the development side. The API Gateway will serve as a temporary defense for any potential threats affecting the API Services, but the development team needs to plan for the remediation of any potential vulnerability.

There are common scenarios where the API Gateway protection capabilities will come into play. Here is a list of the most common protection components in an API Gateway:

- Denial of Service (DoS) Protection
- Web Application Firewall (WAF)
- Trusted IPs (Whitelisting)
- Device Protection

Denial of Service (DoS) Protection

API Gateways provide mechanisms to control how many requests are received by the API Services in a specific timeframe. For example, within an hour the API Gateway can allow a maximum number of requests. For this type of control to be effective, it becomes important to understand the traffic pattern of API Services to benefit from this type of protection without impacting the associated services. Careful evaluation of the traffic patterns will help define the specific metric associated with the requests. These measurements must be collected from the API Service infrastructure.

For example, the Application Server logs will capture every request submitted to the service during normal operating hours. With this information, the API Gateway can be configured to ensure the number of requests aligns with the normal operation of the service. Since there could be variations in traffic, it becomes important to define a buffer for growth and continuously evaluate if the parameters need to be adjusted.

Web Application Firewall (WAF)

API Services will be prone to common threat vectors. SQL Injection attacks and broken authentication are two examples of the most common security vulnerabilities which could be found in API Services. Since development teams will need time to perform code changes, tests and implement any potential remediation to these threats, organizations need temporary protections while those activities are completed. This is where the WAF will come into play.

WAFs provide a temporary capability to thwart attackers from compromising the services through common exploits. Section "API Common Security Threats" in this paper provides an overview of the most common threat vectors covered and protected by the WAF. It becomes important to understand that even though the WAF will provide the protections, development teams will still need to address these exploits in their source code. The implementation of security controls in the SDLC will minimize the probability of exposing the most common threat vectors.

The purpose of the WAF is to provide an additional layer of defense for any escapes from the security controls in the SDLC. The WAF should not be considered the only solution for common threat vectors. This capability offers a complimentary security feature that must be

supported by a secure development environment. Therefore, teams need to integrate the security controls in the process to ensure the risk is minimized of a potential compromise. Please refer to section "Security Controls Needed (SAST, SCA, DAST, IAST) in the SDLC" in this paper for more information about the required security controls in the SDLC.

Trusted IPs (Whitelisting)

By default, API Services could potentially be exposed to the whole organization. For example, an organization could have an API Service providing the bill of materials (BOM) for a piece of hardware. The important question here is, why do we need users in Human Resources (HR) to have access to the service? If the access is not providing any value to their tasks, why is it available? From this standpoint, we can understand how such a stance could enable an attacker to potentially exploit the API Service.

Imagine a scenario where a user asset in HR gets compromised. Because of the lack of controls in terms of who can access the API Service, now the attacker will have a venue which can be used as part of a potential compromise. This is why reducing the attack surface is important. API Gateways provide capabilities to create lists of Trusted IPs to only allow users or assets that have a need to know. This is especially valuable in scenarios where there is system-to-system communication. In those scenarios, there is no need for any users to access the API Services defined to support this type of interaction. Utilizing a Trusted IP or

whitelisting feature will ensure the API Service is accessed and consumed by assets with a need to know.

Device Protection

Just like with Trusted IPs, the API Gateway provides a feature to control which type of device could connect to each API Service. This will guarantee that whenever there are scenarios where the only expectations of a mobile devices is to consume an API Service, any other type of asset is prevented from leveraging the service. This type of control will vary depending on the implementation. Coordination with the API Gateway administrator will be key as only the implementation team will be able to provide guidance in terms of which type of asset is expected to communicate with the API Service. Just like Trusted IPs, this feature will minimize the attack surface against the API Services and will serve as an additional layer in the defense in-depth strategy.

In essence, all the previous protections provided by the API Gateway could be using different approaches. In one of the approaches, the API Gateway may use the definition of rules to control who can leverage the API Services and how. In another approach, the API Gateway may use the evaluation of traffic and signatures to detect potentially malicious threats. Regardless of which approach is provided, organizations must validate the API Gateway is providing enough flexibility to ensure the protection of API Services against common threats to the API infrastructure.

API Access Controls

Access controls in API Gateways provide governance over multiple aspects of the implementation. Whether we need to control who can access API Services, specific methods in the API or who can have access to the API Gateway in the first place, having authorization controls are an essential component of the API Service ecosystem. Because of the number of potential API Services in the infrastructure, it becomes important to organize the APIs using categorization to avoid having to manage APIs individually. The categorization will enable grouping the services in a business logical approach which can then be used when establishing access controls. With this in mind, this paper covers some recommendations on how to use categorization in the "Categorization for the APIs" section.

Many API Gateways control access through the definition of policies. The policies are defined to allow specific group of users to gain access to specific API Services and even functions. Whether we need to control serverless functions in a cloud environment or API Services themselves, the flexibility provided by policies not only make it easy to define the access controls but also easy to manage. Enabling an organized and well-thought design of the categories is going to be an important step here prior to start the definition of the policies. In essence, policies can be defined to provide access controls to:

- API Services
- Resources and the associated methods

- Methods within an API Service
- Additional resources in the infrastructure supporting the API Services

There are instances where the API Gateway can also have access to other important components of the infrastructure such as Secret Management solutions. Access controls can be defined to allow the API Gateway to retrieve specific credentials needed by an API Service to secure sensitive information that otherwise would have to be included in clear text in the configuration files associated with the API Service deployment. This type of access ensures that APIs are serviced with, for example, database connection string credentials during runtime to avoid having to explicitly include them in any configuration file. This approach also supports other important features of secrets management solutions such as enabling automation as well as supporting the lifecycle of secrets (password rotation requirements, revocation, rotation, etc.).

Access controls in API Gateways support the authorization layer. Just like any other type of software, controlling who can access the APIs and its resources as well as who can access resources in the network with the API Services is a key component of reducing the attack surface. Planning and design are key components as the organization needs to plan for the rollout of policies providing governance over the API Services. Leveraging categorization will ensure a standard allocation of access to resources which will support a strong least privilege approach. These concepts will be essential to limit the risk and exposure of organizations.

Compliance and Governance

Regarding API Services compliance, there are several factors that must be considered by the application team, business, and security teams when determining what security controls to apply to APIs and how to properly govern them. Compliance and governance should be driven by business regulations, the frameworks that the business must adhere to, as well as best practices on data protection based on:

1. Who will be consuming the API?
2. How can the API Service be accessed?
3. Where can the API Service be accessed?
4. App/Dev team guidance on how it should be used.
5. App/Dev team API Service documentation.

Answering the previous questions and formulating a cohesive strategy for those concerns are critical components to the foundation/base of your security and compliance strategy. For example, knowing how API Services should be used and having proper documentation provides the structure for auditing APIs and determining when API abuse or misconfiguration have occurred. Furthermore, knowing from where the API Service can be access and who has access are just as critical, for example knowing if API Services require an API Gateway to access or if APIs can be accessed externally or by circumventing security controls. Having

these foundational items documented and enforced with both technical and administrative controls will allow for better API Service compliance and governance on the use and consumption of APIs.

With regard to the API Services' compliance to frameworks, the most popular source mentioned in the paper is the OWASP Top 10 API 2019. This is a good starting place for API Services hardening (posture management) but isn't the only credible source; companies should look at not only compliance with API Service security standards but with other security regulations as a whole, such as NIST (800-53 or 800-171) or ITAR, etc.

One of the biggest goals for API Services compliance and governance is the prevention of data leaks due to exposed APIs. This is done in many ways and as described in this paper is a layered approach, but one of the most important aspects is enforcement of access/data rules and the monitoring and analytics of those rules; and in reality the monitoring of API Service consumption as a whole. This can come in many forms but most use SIEM tools or purpose-built API Service security tools to perform anomaly detection and behavior analysis.

Another key aspect of this is the proper management of API versions and the deprecation of older version of APIs, this is critical for home grown/internally developed APIs where older version might simply be forgotten or lost due to developer turnover or poor documentation,

so to reiterate two of the most important factors for API Service compliance and governance is the discovery/inventory of APIs and proper documentation for APIs within your environment. There are a number of critical questions that can help determine the level of compliance and how metrics can be used to better support implementation of API Services protection and detection methods.

Metrics/Questions to Assess Compliance and Governance:

1. Is there a full inventory of API Services documented? Is there a discovery method/plan in place?
2. Are the API Services properly categorized?
3. Are the known APIs Services documented? What is the process for documenting discovered/rogue APIs?
4. What is the internal hardening standard based on and how often is it reassessed?
5. What regulations apply to the business and API Services?
6. What is the data protection/security standard system?
7. Are there effective management and controls in place for externally accessible API Services?
8. What monitoring and analytics for API Services consumption are in place?
9. Are external entities allowed access to API Services ?
10. Can DAST scans be performed against API Services?
11. Can passive monitoring of API Services traffic be performed?

12. What tools or SIEM alerts have been (or need to be) developed for advanced misconfiguration and anomaly detection?

This is not a comprehensive list but a good starting list of questions every mature application security program should ask when developing compliance, governance, and general security strategy, encompassing API security.

API Key Management

Even though API Keys may not be the most secure approach to leverage API Services, there are specific scenarios under which using them may be useful. Consider the use case where traffic is going to come from an anonymous entity through automation, some control is needed in terms of how many times the service gets called or identify how often the service is being used for some specific purpose (billing, logs, etc.). If any of those scenarios are required, then API keys can be used but the keys must be protected. The use of API keys can authorize an endpoint to be able to call the service. This is especially true in instances when automation is implemented. In order for the automation to be effective, human friction needs to be removed, the use of API keys becomes a perfect fit for the automation scenario.

There are some basic rules which should be considered whenever teams are trying to secure the API keys. The basic rules can be summarized in the following key points:

- Key Protection
- Key Rotation
- Key Revocation

Key Protection

API keys should be treated like passwords. After all, API Keys provide a mechanism to access the backend and may expose sensitive processes or data. Therefore, API Keys should never be stored in clear text. Teams should avoid the urge to use properties or configuration files to store them. The preferred method of protection will be the use of a Secrets Management solution. If a Secret Management solution is not an option, encryption should help provide the level of confidentiality needed. API Keys can be stolen, therefore it is important to control how many times the API Key can be used concurrently or utilize rate limit. Without a rate limit, multiple endpoints could use the same API Key. In case of a compromise, if rate limit protection is not being used, an attacker could potentially use the key multiple times creating bottlenecks in the processing and impacting the normal operation of the API Service. API Gateways provide good control over the rate limit of API Services. See "API Gateway" section in this paper for more information.

Another important protection consideration is to limit what can be done with the API Key. It is common for teams to issue one single key that is configured to have unlimited access. This

creates the scenario where any compromise of the key will open the door for an attacker to have a non-restricted attack surface. Ensuring the key is only used under a specific set of access controls rules will guarantee organizations will have a limited attack surface. Having API Keys with extremely wide access controls may provide a venue for attackers to be able to leverage any security threat and compromise the underlying system. Furthermore, organizations should avoid using a single key for multiple purposes. Employing a separation of concerns approach will dictate the need to create multiple keys for multiple purposes. This will limit the attack surface for an attacker if any of those keys are compromised.

Key Rotation

Despite the implementation of security procedures to manage API Keys, there is always the risk of leaked keys. To minimize the window of exposure associated with those events, API Keys need to be rotated. The rotation period must be set depending on several factors like compliance, exposure, data sensitivity, etc. For example, a platform being evaluated against the Cybersecurity Maturity Model Certification (CMMC) requirements for secure usage of API Keys would be expected to rotate keys every 90 days to comply. If the data is not sensitive, the API Services are intended to be consumed by the business, and they are not internet-facing, a year could be used as the standard for API Key rotation. Each organization will need to evaluate the risk ranking of the API Services to decide which rotation period should be exercised.

Key Revocation

Key revocation involves the ability to remove a key from an operational state. There are several scenarios under which API Keys could be revoked. The following list provides several valid scenarios for key revocation:

- Compromise – key disclosed to an unauthorized entity
- Administrative – key owner left the organization or API Service has been decommissioned and the key is not used in any other service

In the case of a compromise, revoking an API Key requires a methodical process. There are three steps in the revocation process:

1. An assessment for the need for the revocation is conducted. Since the API Key would be very likely used by different systems, even in automation activities, communicating with the owners of such initiatives will be key. During the revocation assessment, it will be extremely important to identify who must be notified regarding the API Key revocation. Careful coordination should be exercised to avoid service disruption.

2. The new API Key is created. Once the new API Key gets created, coordination to share the new API Key to those entities affected should be conducted using a secure channel. Avoid sending the API Key in clear text. Encryption is recommended to safeguard the confidentiality and integrity of the key. Likewise, the team receiving the key must ensure it is protected and should avoid storing the API key in clear text. Here, the use of a Hardware Security Module (HSM) or encryption is recommended to ensure no one can steal the key and use it.

3. Finally, the API Key revocation is executed. At this point, the coordination needed to execute a revocation (which will impact any endpoint consuming the service) has been completed and communicated. A new key could be issued at this step if the organization needs to restore the access to an API Service.

Following these main three considerations will ensure API Keys are used and protected according to security best practices. One final point to consider is associated with whether users or systems will be using the API keys. If user authentication is needed in the interaction with the API Services, then API keys should not be used. There are other authentication mechanisms more effective and secure which could be used whenever users are the primary target of the API Services usage. Some of these authentication mechanisms are covered as part of the "API Gateway" section of this paper.

External API Services Security Concerns

External threats to APIs come in many forms and are not necessarily new kinds of attacks. API Service attack vectors could also be used to against web applications and other externally facing services. Bad actors take advantage of misconfigured or weak API sessions which can lead to data loss or help establish a foothold on one or more systems potentially allowing movement laterally to other systems. Understanding each form of attack and addressing the risk through a correct configuration or mitigation process will go a long way in reducing the attack surface.

Some of the potential threat vectors will come from what the OWASP organization has identified as the Top 10 API security threats. These are covered in "API Common Security Threats" section which provides in-depth details regarding each (OWASP, n.d.). There are several best practices which will help organizations secure their access to these external APIs and minimize the associated risk. the next several sections provide a quick glance at the options available. Table 7 provides a summary of the most common concerns associated with external API Services.

Table 7: External API Service Security Concerns

Main concern	Guideline
Understand your API technologies and environment	Organizations need to understand what technologies are in use and how to configure them securely using industry guidance. Moreover, inventory, tracking and logging API use is needed to gain visibility into daily operations.
Governance around adoption and configuration	Develop a governance model that includes policies, guidance, outlines risk and establishes best practices.
Use APIs as necessary to achieve the desired state	While an organization might be quick to just connect an API, evaluate all options to ensure the use of an API is the best course of action and then follow guidance for securing and handling them.
Identify vulnerabilities	Since APIs fall under the "software" category, it is important to track potential vulnerabilities either in the languages used, or the code itself. APIs need to be treated like any other software product.
Leverage OAuth	OAuth falls under authentication and authorization and is used to make secure connections to API. This type of access helps to protect user credentials.
Use tokens	Tokens can be used to establish a trusted identity and access to services.
Encrypt data	Encryption needs to occur at all levels here. Any sensitive data such as sensitive personal data or SPI needs to be protected both at rest and in transit. Transport Layer Security (TLS) is normally the method by which the session data is protected.
Use rate limiting and throttling	Rate limiting and throttling the connections can help reduce attacks around DDoS.
API Gateway and Web Application Firewall (WAF)	API Gateways are used to receive requests for API Services and ensure these connections reach their targeted endpoints. WAFs are used to decouple the session and inspect traffic as it passes through. While neither of these devices are needed to allow APIs to function, they provide

	great visibility and other security metrics for protecting against malicious activity.
Service mesh	Similar to API Gateways, a Service mesh is used to manage traffic between microservices and associated elements to produce the transactions. One major advantage to using a Service mesh is that it can automatically determine the microservices in use. Without a service such as this, the logic for these types of connections would need to be coded into the application. A Service mesh is often used in conjunction with an API Gateway (Tozzi, 2021).
Implement a Zero-Trust Model (ZTM)	Zero trust implies that nothing or no one is trusted and therefore security controls are applied to specific assets. This type of model should be applied across all organizations and not just APIs.
Parameter Validation	Data streams used by the API need to be validated to ensure there's no risk to the configuration. This is handled by what's known as Schema Validation and will vary by configuration depending on the technology used.

Additional API Threats

Similar to governance, establishing threat models will help the organization identify risks with the use of APIs. These types of models are usually conducted upfront and can help assess where security controls need to be placed. These risks can come in many forms and may originate from the internal organization's lack of a strong security posture, other first and third parties, the client who is the actual user of the API, or the consumer. Controls around each of the APIs need to be documented and in pace. If the organization has no strong security measures in place during the coding of their APIs, this can have an immediate

impact on exposure. Whether the risk is from poor coding practices or inadequate security controls such as strong authentication or encryption, the increased risk of API compromise is almost assured.

It is important to understand where and how APIs are being used in the organization. Not only does this map back to the business side for ongoing costs incurred, but it allows for understanding the actual uses, and deviations which may lead to unscrupulous activities with the APIs. Identification and inventory of the organization's APIs will not only help the business understand the costs associated with the use of APIs but reduce the risk exposure by knowing where the APIs exist and ensuring they follow the security and deployment guidance previously laid out by the organization. Accountability as to who is responsible for what activities with regards to APIs need to be well understood and demonstrates maturity for organizational support of the company's API inventory. This can be accomplished with a simple accountability matrix drawn up to identify all parties involved with API development and support.

API Code and Security Testing

It is imperative to ensure testing of the organization's APIs take place to understand any risk exposure or security controls needed. For development, this might begin with the OWASP Top 10 and any applicable threat vector from the SANS Top 25 (SANS, n.d.), but a deeper

dive would seek to test also any threat vector not listed in these two lists. First and foremost, does the API perform the way it was coded? Is validation coded so that if unexpected inputs were made the API predictably responds according to its logic model?. From a performance perspective, can the API handle the load? Recall, one of the tenants of the CIA Triangle is Availability. Does the API function technologically agnostic? These and other test checks will determine if the API is built in a manner that reflects best security coding practices. It has been noted that the SOAP protocol where XML is used, has many known security risks that need to be resolved before implementation.

System security checks should be conducted to ensure the best practices for controls are in place. Authentication and validation (Data, Users and Certificate), as well as encryption, should be top priorities for testing and protecting the API. Moreover, APIs should be built with Least Privilege concept in mind with only the necessary data to be delivered to satisfy the business logic. External controls such as API Gateways, Firewalls, inventory solutions, etc., need to be reviewed to ensure they are performing in the manner they were deployed for. In like manner, monitoring and logging with data feeds to the organization SEIM system should be validated and understood before going into production with the API.

Conclusion

The use of API Services supports better capabilities in newer architectures such as microservices and containers. The industry is moving at an accelerated pace to adopt API Services architectures to provide more flexibility and better features to end-users and customers. Despite all the benefits provided by them, organizations need to ensure API Services are not weaponized by attackers and used as vehicles to compromise the technology infrastructure. Security controls in the SDLC must be implemented to help identify security vulnerabilities in API Services using an early detection approach and enable development teams to remediate these threats before rolling these services to a production environment. This paper introduced the most pressing security threats impacting API Services as presented by the OWASP organization. Understanding the threats will enable software engineers to enforce a more defensive posture during the software development process.

An important step in the API Service strategy involves the discovery process. Having a discovery process will ensure easy detection for not only known API services but also rogue ones not captured by the security strategy. In tandem, the understanding of their use is also a key component which will help delineate not only the governance needed but also how to control access to them. Using an API Service classification strategy will facilitate the

implementation of policy-based decisions to group them depending on multiple key factors such as data sensitivity, business impact, deployment environment, etc.

Establishing standard governance over API Services will ensure a comprehensive adoption of security rules and controls across the board. The use of an API Gateway provides the means to support the standardization and governance needed. API Gateways provide meaningful detection, protection, and authorization controls using a centralized approach. Without them, the organization will be required to secure each API Service endpoint individually, which will increase the management overhead and the potential risk associated with the lack of a standard security posture.

Recognizing the different scenarios available for the consumption of API Services is very important and the organization must plan to establish a process to manage it. System-to-system communication will require different protection rules than direct end-user consumption of API Services. Understanding these differences will help an organization establish the right approach and ensure best practices are followed. Organizations will not always have control over the code used to develop some of the software used to implement the API Services. There are cases where external or third-party services will be consumed which will limit our ability to control remediation for vulnerabilities in code we do not own. Even though we may have little latitude in terms of the remediation of potential threats in

these API Services code, it is important to establish a plan to protect their use and minimize the risk they could pose to the organization.

References

Github. (2022, April 1). *The Registry API*. https://apigee.github.io/registry/

OWASP. (n.d.). *Owasp API Security Project*. OWASP API Security - Top 10 | OWASP

SANS. (n.d.). SANS Top 25. https://www.sans.org/top25-software-errors/

Tozzi, C. (2021, Sep 15). *Service mesh vs. API Gateway: Where, why and how to use them*. https://www.techtarget.com/searchapparchitecture/tip/Service-mesh-vs-API-gateway-Where-why-and-how-to-use-them