Application Programming Interface (API) Technical Guidance

Minimum Viable Capability Release (MVCR) 1
July 2024

Office of Systems Engineering and Architecture

Office of the Under Secretary of Defense
for Research and Engineering

Washington, D.C.

Application Programming Interface (API) Technical Guidance

Minimum Viable Capability Release (MVCR) 1, July 2024

Office of Systems Engineering and Architecture
Office of the Under Secretary of Defense for Research and Engineering
3030 Defense Pentagon
Washington, DC 20301
osd-sea@mail.mil | Attn: Software Engineering
https://www.cto.mil/sea

Approved by
Principal Deputy Executive Director, Systems Engineering and Architecture
July 2024

API Technical Guidance Change Record

Date	Change	Rationale
October 2023	Initial release.	
July 2024	New Sections 5 and 6. API Team review and updates.	Minimum Viable Capability Release (MVCR)-1

This page is intentionally blank.

Contents

Figures

Tables

1 Introduction

In the rapidly evolving landscape of modern warfare, the U.S. Department of Defense (DoD) relies on advancements in technology to maintain a competitive edge in joint warfare capabilities. Central to these advancements are software application programming interfaces (APIs). An API is "a system access point … accessible from application programs . . . to provide well-defined functionality" (NIST SP 1800-21). APIs promote interoperability, security, and scalability.

Interoperability, "the ability to act together coherently, effectively, and efficiently to achieve tactical, operational, and strategic objectives" (CJCSI RSI 2019), is the priority of the Joint Staff (Brady and Dianic 2022). Interoperability is crucial to modern software, joint warfighting, artificial intelligence (AI) superiority, and achieving the Deputy Secretary of Defense Data Decrees (DepSecDef 2021).

APIs are essential to interoperability (Brady and Dianic 2022). APIs facilitate data sharing, collaboration, and the seamless integration of systems and capabilities across different branches and units within the Department and with allies (e.g., NATO). Other key concepts of APIs include sensor fusion and Internet of military things (IoMT) operational integration, emerging technology adoption, rapid prototyping and experimentation, ecosystem development and innovation, and protection of critical and emerging technologies.

APIs facilitate seamless communication among diverse software systems and enable the creation of sophisticated, integrated applications. Composability of APIs allows numerous capabilities to be aggregated rapidly into new and distinct capabilities.

The Office of the Under Secretary of Defense for Research and Engineering (OUSD(R&E)) office of Systems Engineering and Architecture (SE&A) led the development of this document in collaboration with the OUSD for Acquisition and Sustainment (A&S) and a team involving Joint All Domain Command and Control (JADC2) and more than 20 DoD Components.

1.1 Purpose and Scope

This guide provides an overview of API concepts in software development. This follow-on minimum viable capability release (MVCR)-1 adds content to the initial minimum viable product (MVP), and adds DevSecOps and Test sections to the initial MVP sections on governance, cybersecurity, zero trust (ZT), zero trust architecture (ZTA), and API design and implementation principles. Future versions will include additional topics e.g., real-time APIs, software development kits (SDK) and libraries, secure programming and error handling, performance optimization, and scalability, and more.

This document includes use cases, lessons learned, and best practices from DoD and industry. Although other guides exist, this guide emphasizes the importance of enhancing and advancing the DoD warfighting capabilities of the near future to support the Combined JADC2 (CJADC2) vision and to secure information interoperability across the DoD. (See also Appendix C: API Security Challenges for more detail about the CJADC2 vision.) The guide describes an API framework to help programs define their technical baseline for delivering future systems that support the DoD enterprise and warfighter mission requirements.

Figure 1-1 illustrates the scope of APIs covered in this guidance from a system perspective.

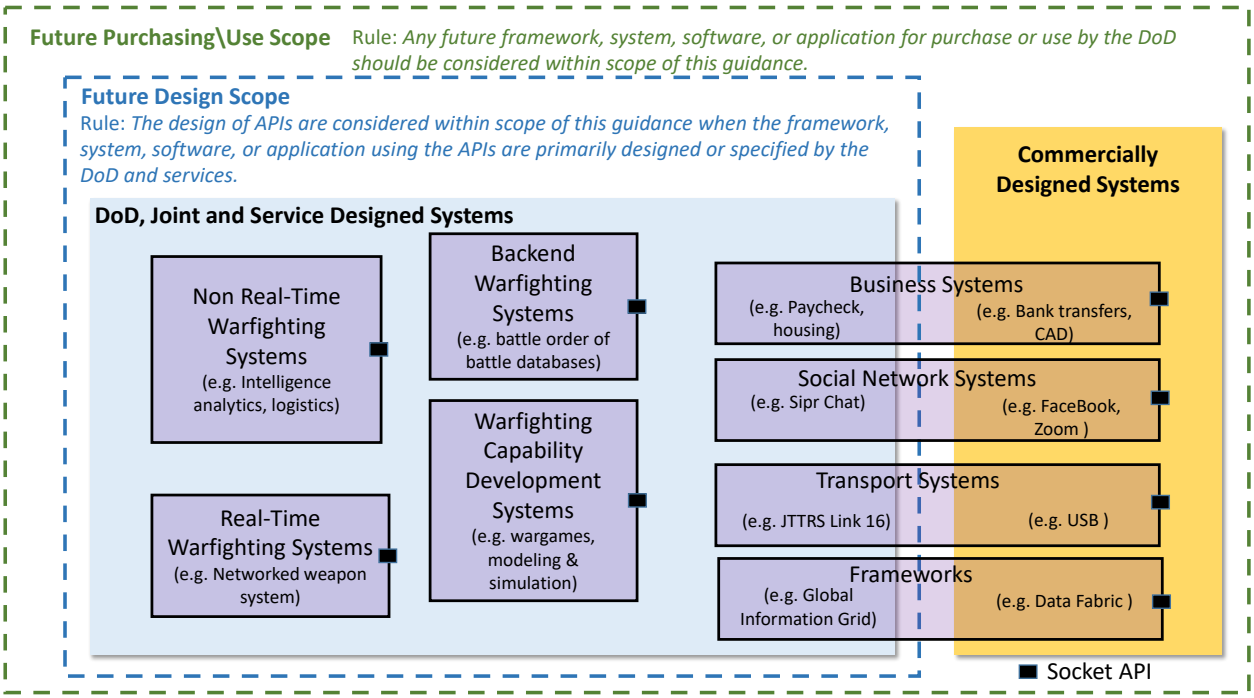


Figure 1-1. API Context Diagram from a System Perspective

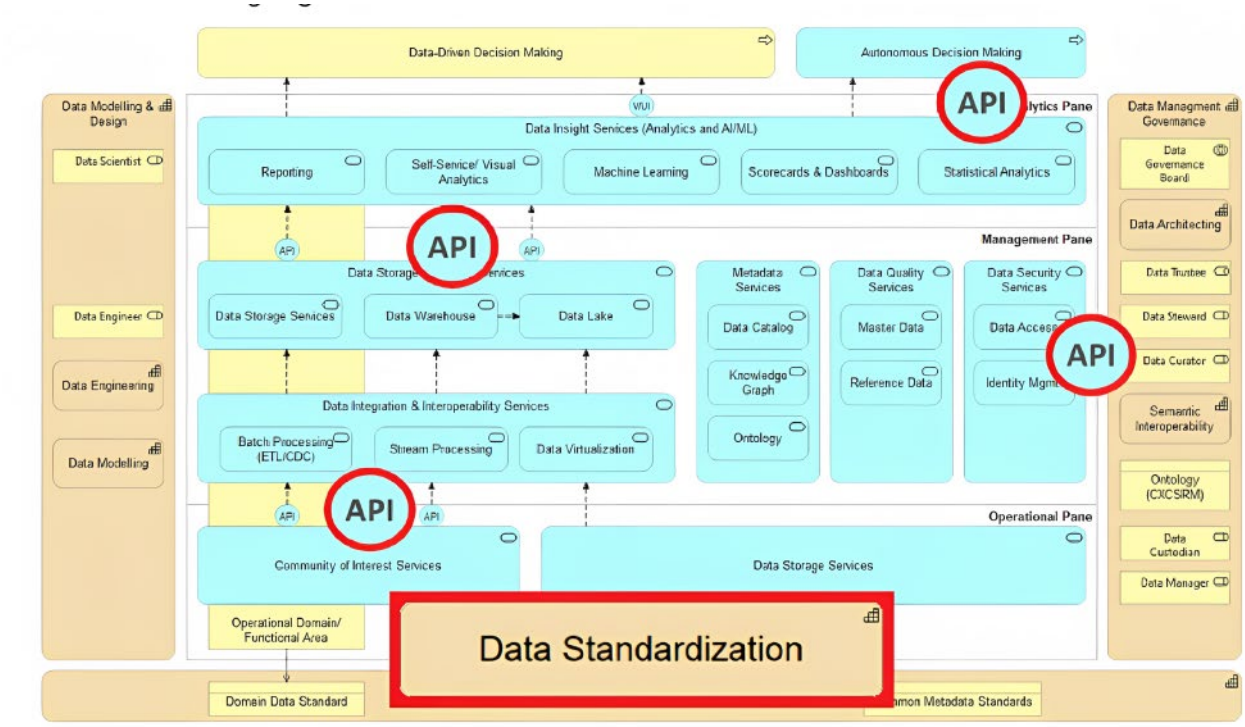
This figure views an API as a socket connection between two systems. In general, a socket (e.g., Web, Berkley, Windows, Unix, Linux, Java) is an abstract representation for the local endpoint of a network communications path. This perspective dichotomizes API ecosystems into those designed for the DoD and commercial industry.

Four types of DoD-specific warfighting systems are non-real-time (e.g., intelligence analytics, logistics); real-time (e.g., networked weapons); back-end (e.g., order of battle); and capability development (e.g., wargames, modeling and simulation); however, the commercial API ecosystems such as business systems (e.g., paychecks); social networks (e.g., SIPR chat, other ChatOps); transport systems (e.g., Link 16); or other frameworks (e.g., Global Information Grid) are also partially in scope. Each such system contains one or more open or proprietary API

socket interfaces connecting with other systems. The future design scope of APIs includes the four DoD systems and, in part, the four commercial systems.

In the future, the DoD should maximize APIs developed using open standards and minimize the proprietary data interfaces the governance does not control. These requirements must be clearly addressed within the acquisition processes and contracting language to avoid proprietary, contractor-specific, vendor lock-in integrations throughout the program lifecycle.

The scope of APIs also can be seen from a data perspective as shown in Figure 1-2.



Source: (NATO 2023)

Figure 1-2. API Context Diagram from a Data Perspective

This figure views an API as an automated data standard between two services. In general, a data standard is any documented agreement on the representation, format, definition, structuring, tagging, transmission, manipulation, use, and management of data (EPA 2023). An automated data standard or API can reside at various levels including between autonomous decision-making and data insight/analytic services (e.g., reporting, machine learning, statistical analysis); analytics services and storage services (e.g., data warehouse, data lake); data integration and interoperability services (e.g., batch or stream processing or data visualization) and community of interest services; or data management and governance and the management plane (e.g., data quality and security). Thus, any API in use by, designed by, or specified by the DoD, Joint, or Services is considered within scope of this guidance.

The following items will be reserved for a future release:

- Real-Time APIs
- Software Development Kits (SDK) and Libraries
- Secure Programming and Error Handling
- Performance Optimization and Scalability

1.2 Intended Audience

This document is intended for a range of stakeholders involved in the design, development, deployment, and management of APIs across the DoD, industry, and academia, including the following:

- Architects, designers, developers, and testers responsible for designing and implementing APIs and data frameworks
- Program Managers (PMs) responsible for overseeing API development and deployment
- Security professionals responsible for ensuring the security and compliance of APIs
- Policy professionals responsible for maintaining policy for the DoD
- Acquisition professionals responsible for creating acquisition guides, pathways, and policy
- Operations and support staff responsible for maintaining and monitoring APIs

1.3 Document Relationships

Figure 1-3 shows other guidance documents related to this topic and their relationships.

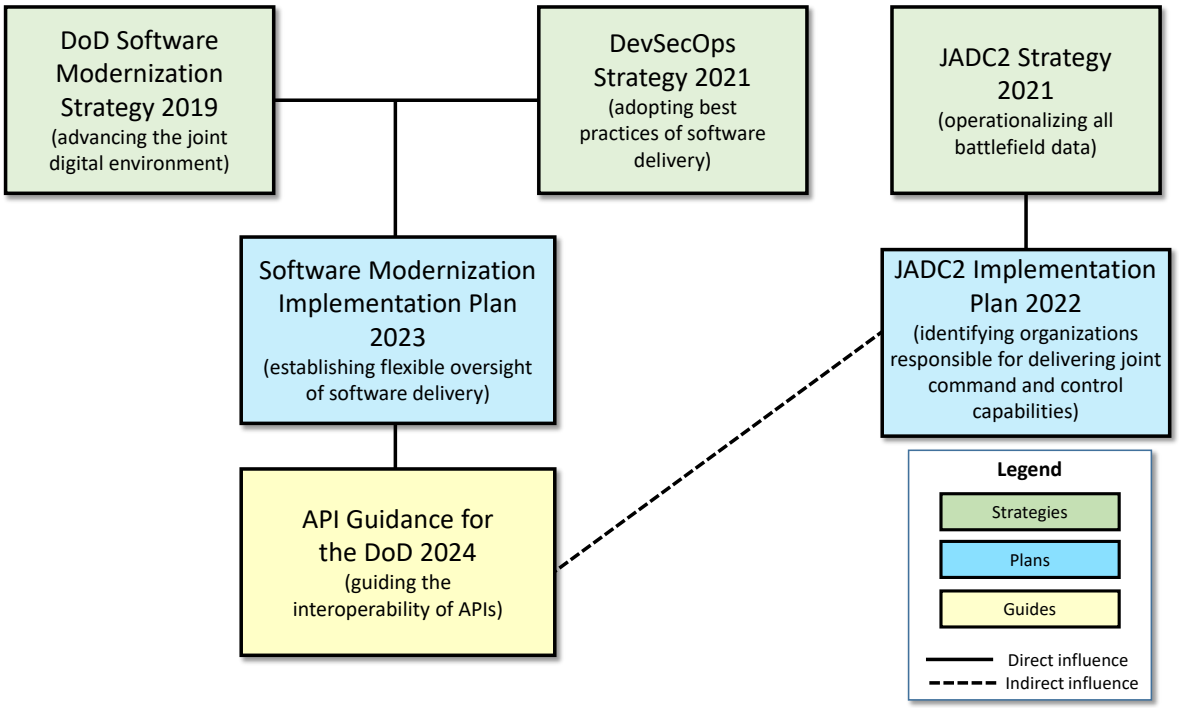


Figure 1-3. Document Relationships

This guide is primarily related to three DoD strategies for advancing the joint digital environment: DoD Software Modernization Strategy (DepSecDef 2021), the DevSecOps Strategy (DoD DevSecOps Strategy 2021), and the JADC2 Strategy (Hoehn 2022).

The guide discusses adopting best practices (DoD CIO Library 2023) and operationalizing battlefield data (Hoehn 2022). The guide is influenced directly by the Software Modernization Implementation Plan (DoD CIO 2023) and the JADC2 Implementation Plan (DoD 2022) and indirectly by close collaboration with the Joint Staff.

1.4 DoD Landscape

Brady and Dianic (2022) observed that "in contrast to commercial industry and modern web economy. . . . DoD lacks a coherent API ecosystem." They note that 21st century businesses know investing in an API strategy pays significant dividends, and APIs or exposed interfaces (with controlled access) promote interoperability, security, and scalability.

The DevSecOps strategy (DoD CIO Library 2023) also notes the challenge of having to rely on few vendors for certain interfaces: "DoD must acknowledge a lock-in posture; recognizing vendor lock-in, *and* recognizing product, version, architecture, platform, skills, legal, and mental lock-in also exist" (DoD DevSecOps Strategy 2021).

This document considers the following DoD API goals:

- Combined Joint All-Domain Command and Control (CJADC2)
- Modular Open Systems Approach (MOSA)
- DoD Data Strategy (2020) VAULTIS Goals which must be achieved to become a data-centric DoD.
    - Make Data Visible - Consumers can locate the needed data.
    - Make Data Accessible - Consumers can retrieve the data.
    - Make Data Understandable - Consumers can recognize the content, context, and applicability.
    - Make Data Linked - Consumers can exploit data elements through innate relationships.
    - Make Data Trustworthy - Consumers can be confident in all aspects of data for decision-making.
    - Make Data Interoperable - Consumers have a common representation/ comprehension of data.
    - Make Data Secure - Consumer data is protected from unauthorized use/manipulation.

1.4.1 API System Development Paradigm

An “application program” is a software system “implemented to satisfy a particular set of requirements” (NIST SP 1800-21). APIs help organizations “connect the many different application programs used in day-to-day operations. For developers, APIs provide communication between application programs, simplifying their integration” (IBM API 2023).

An application program resides on a host hardware system, which may be part of a larger system, which in turn may be part of a larger system of systems (SoS) as shown in Figure 1-4. The application program calling the API can be entirely machine code or could have a user interface.

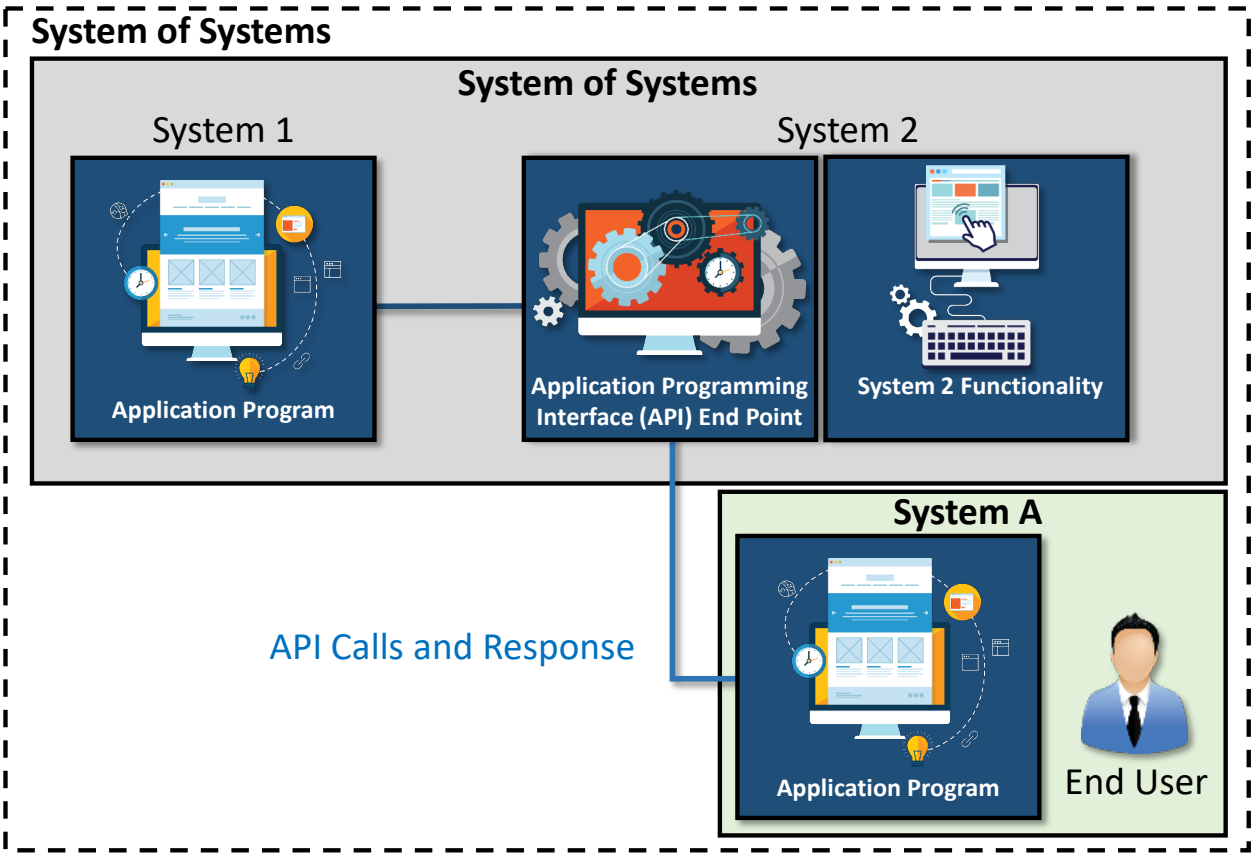


Figure 1-4. APIs in a System of System Context

An API may be used in or developed for a single system or for an SoS environment. Within a single system, the API may meet the same needs as the application program. In a broader SoS context, the same API may or may not meet the needs of external systems and use cases.

For DoD, the API challenge is that each system is acquired via contracts from a specific commercial industry, military industry, or government provider. APIs and application programs in these systems may be designed by a single provider or alternatively by a consortium of providers and stakeholders. These designs should follow guidance from section 1.1 above, focusing on open standards-based APIs; however, the actual APIs and application programs implemented into code by providers are for a specific system or set of systems.

A crucial influence on the design of the APIs is the overall development environment in which they are used. Principles of security, trust, dependencies, test, and production all must be considered (see Section 4).

1.4.2 Interoperability

Interoperability is referenced in many contexts and forms, so establishing where APIs reside among the different concepts is worthwhile. Figure 1-5 shows how the common terms relate to one another.

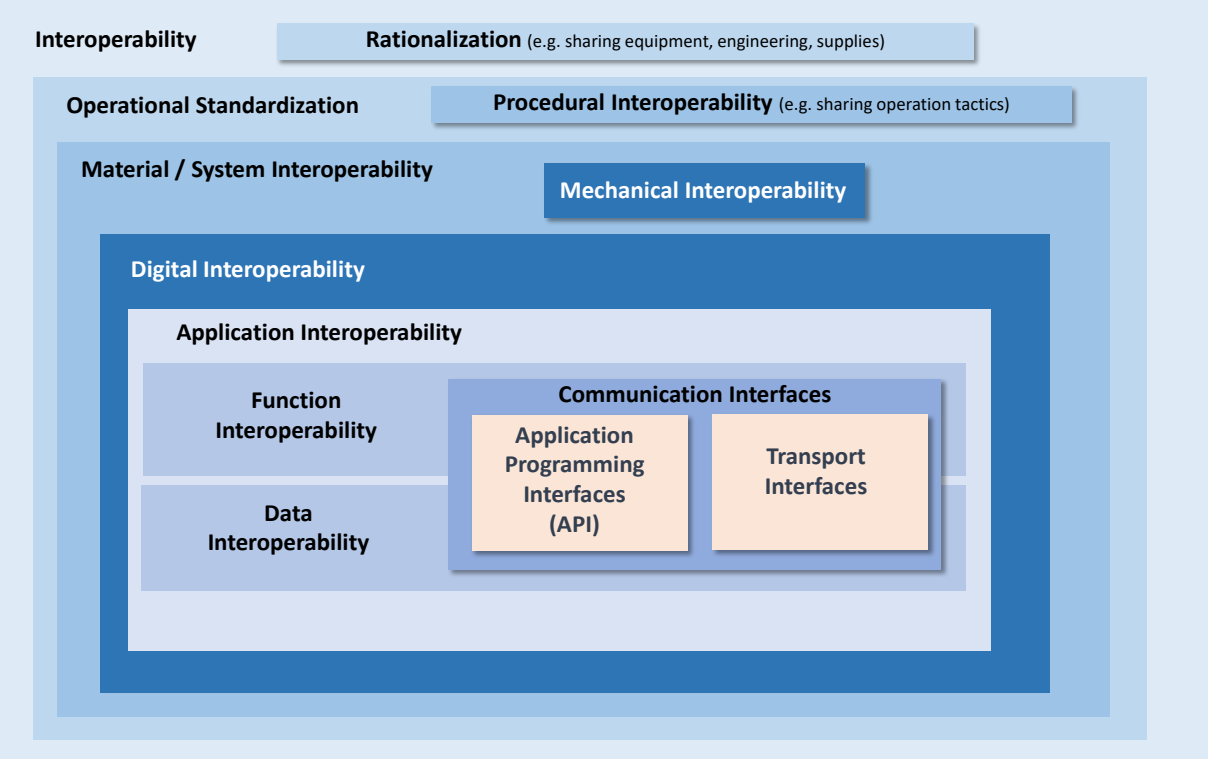
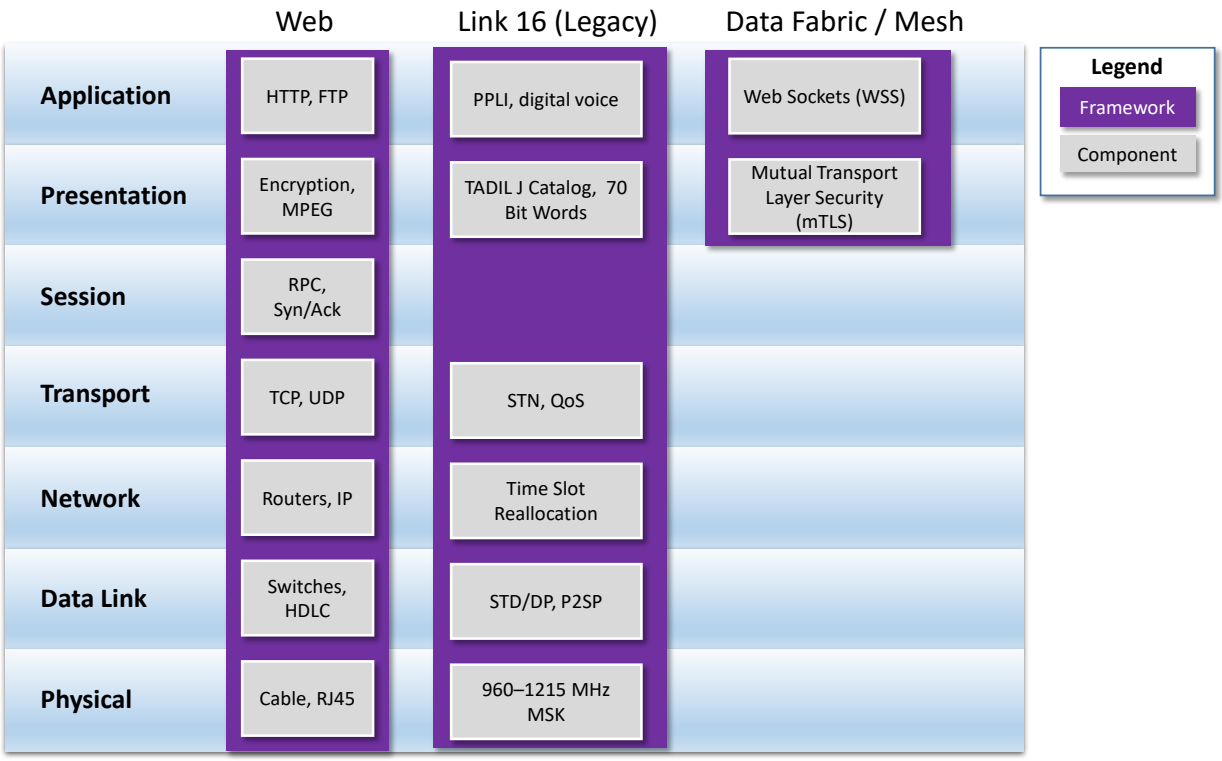


Figure 1-5. Interoperability Concepts and their Relationships

Data strategies, architectures and frameworks are an integral part of interoperability and often use APIs. APIs are used in conjunction with broader frameworks, which could be called models, ecosystems, capabilities, protocols, and messaging standards (ISO/IEC 7498-1 1994) (Computer Networking Notes 2023). Figure 1-6 shows some examples.



Source: ISO/IEC 7498-1:1994; Computer Networking Notes 2023.

Figure 1-6. Examples of Frameworks, Capabilities, Systems, and Protocols

APIs are a form of a message with a certain protocol, often partitioned by the Open System Interconnection (OSI) model (Computer Networking Notes 2023), but not all messages are APIs, as shown in Figure 1-7. Some legacy application programs and systems communicate with each other where the messages are predetermined, and no programming interface is needed (e.g., Link 16). This guide covers all future messaging systems.

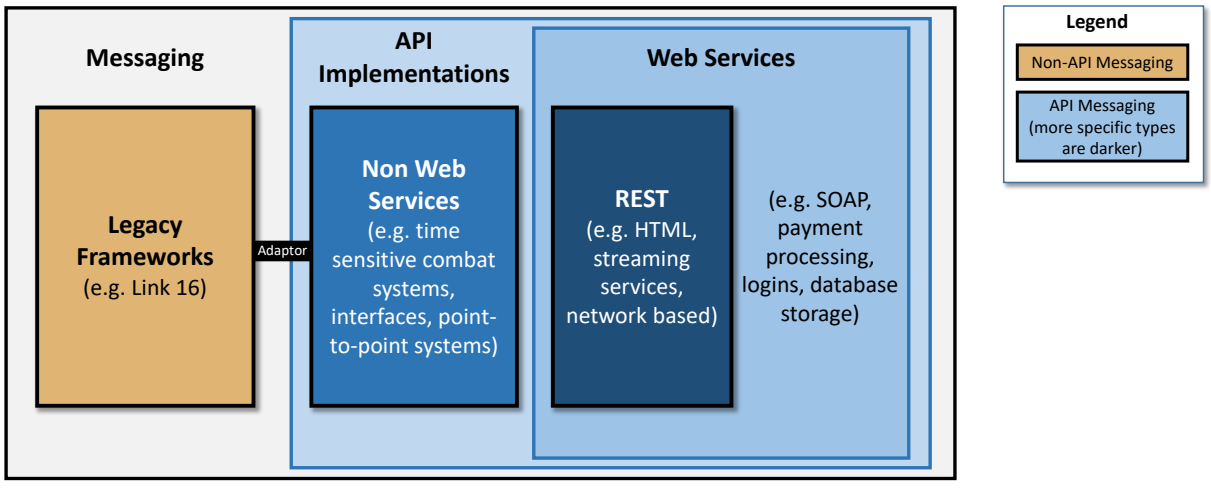


Figure 1-7. Messaging Types

1.4.3 Legacy Systems

The DoD has many legacy systems that are costly and challenging to maintain. Updating these systems with APIs can extend their life. Complete replacement of the systems at once may be costly and time consuming, so some programs have discovered inexpensive and creative techniques to integrate APIs with these legacy systems, ultimately improving capabilities, lowering risk of failure, and extending the life span into the future.

In the objective future state, software processes will start with a discovery phase to identify the existing APIs relevant to the application, their definition, and access pathway. Some new applications will require APIs to be extended, or to be made more available, or new APIs will be needed and added. To reach a more desirable end state, an incremental phased change approach will most likely be needed to employ applications and APIs (including the deprecation of older API versions). This effort will take time and sustained enterprise governance.

Although cost and schedule are key considerations for sustaining legacy systems, any new APIs should follow the guidance in section 1.1 to leverage open standards-based APIs.

1.4.4 Other API Terms

Figure 1-8 shows other terms the community may be familiar with that relate to the terms in this guide. The figure depicts the user (API consumer) connecting to the product (API provider) over a network (transport) to consume data and information services.

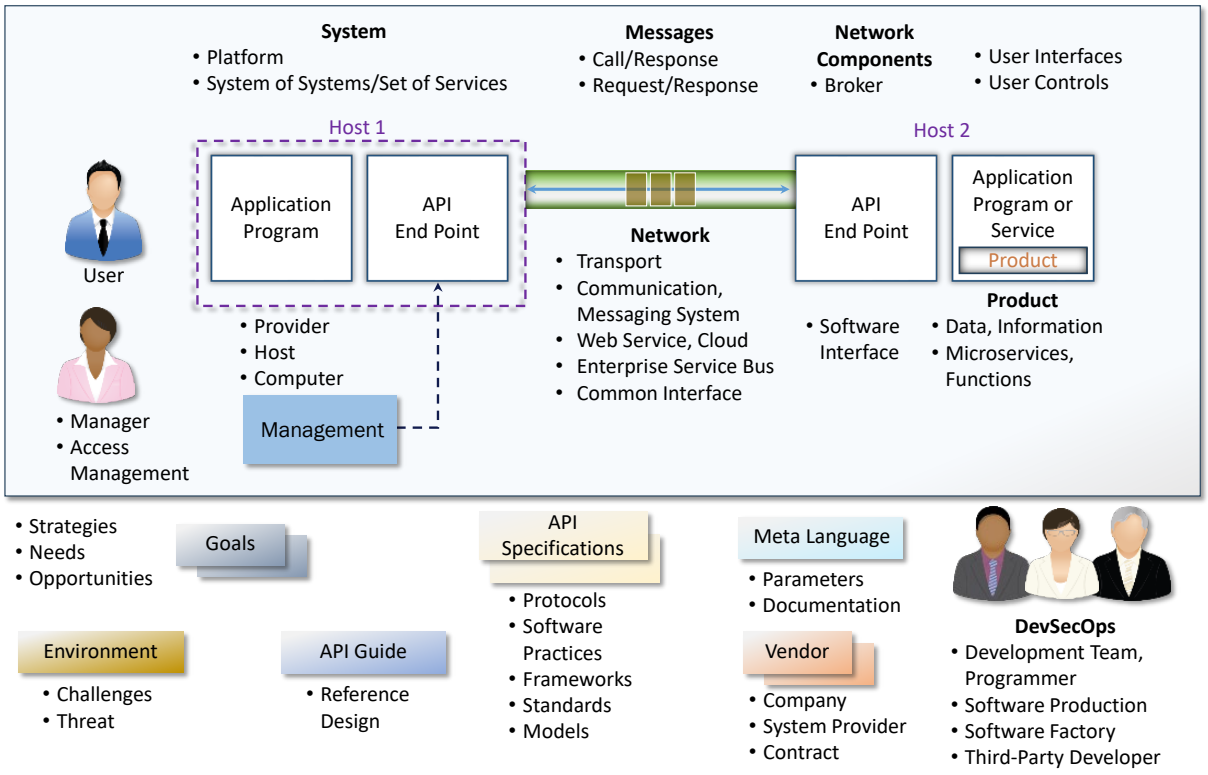


Figure 1-8. API Terms and Relationships

In terms of the DoD, the user might be a soldier OCONUS in theater using their Android Tactical Assault Kit (ATAK) APIs via a handheld android or ruggedized laptop device to access various data services (e.g., Blue Force Tracking, Red Force Tracking, weather, or terrain topology) via the Joint Operational Edge (JOE) cloud to support critical situational awareness (SA) information needed for their mission.

The manager in the diagram (bottom left) could also be accessing management services via the JOE from a CONUS-based SIPRNet system at the Pentagon or other DoD facility to evaluate system metrics and operational status of the family of systems hosting the ATAK APIs (product) at the edge.

2 API Project Governance

This section discusses how DoD API project organizations should create a governance process to guide development, implementation, and updates to their unique API ecosystem or framework. A DoD API project is any new or existing DoD-funded project that is creating APIs or updating their API. These projects could be system development programs for which APIs are but one aspect of the system, or the projects could focus on interoperability standards or API frameworks. All these use cases will be referred to in this guidance as "API projects."

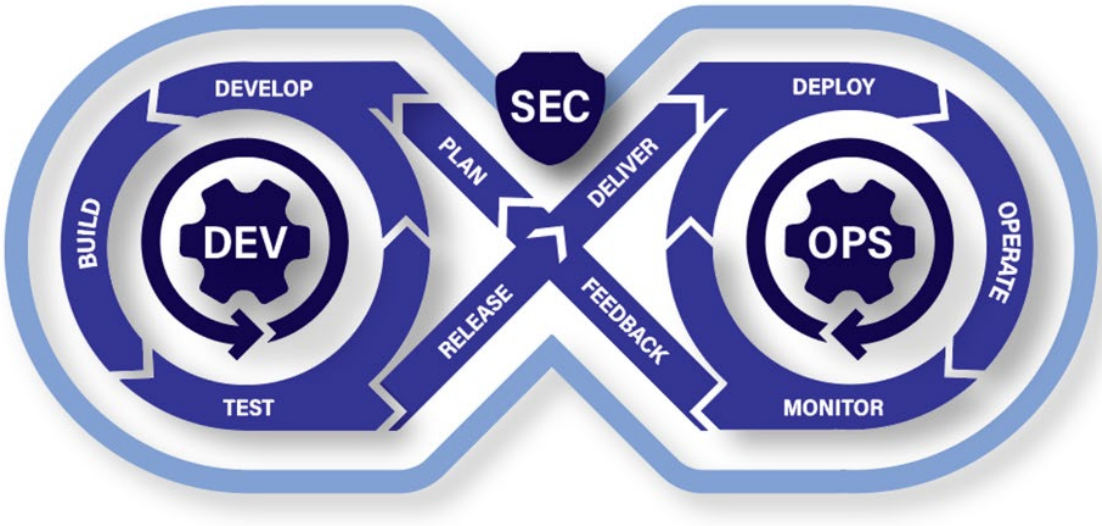
An API project governance model is the application of rules, policies, and standards to the project API ecosystem. Governance includes how the API project organization should share, encourage adaptation, administer, adjudicate, and update the API in support of both internal and external stakeholders. The API project governance process described in this section will evolve as technologies, standards, best practices, and organizations evolve, but the process focuses on open and fair governance, not specific technical recommendations. The guidance does not mean one size fits all. Developers must be free to implement APIs as needed for mission requirements. A governance process and enforcement can ensure the API follows best practice and demonstrates high-quality attributes, resulting in the benefits noted in Table 2-1.

Table 2-1. High-Quality API Attributes and Benefits

Attribute	Benefit
Reusable	Can reuse existing components. Developers need to build components only once and will not end up duplicating code. They can spend more time on tasks that benefit the business, like building new services.
Reliable	The APIs reliably are available and function as documented.
Interoperable	Can be used with approval in all use scenarios by applications that would benefit operations.
Discoverable	Developers can easily find existing API artifacts and reuse them in future designs.
Scalable	Can have small or large number of elements and the APIs can serve many diverse users.
Consistent	API remains consistent even when implemented by different developers and across the entire DoD solution space.
Easy to Use	The API is easy to understand and implement in many diverse use cases.
Clear	The API vision, design, and documentation are clear. Helps keep everyone involved in the API program. Misunderstandings about API goals or designs can cause API programs to fail.
Secure	Security is built into the foundation of the API development and deployments. API interfaces includes classification metadata support (NSA Access rights and handling, information security metadata, NSA Guidance for implementation of REST – in DISR).
Compliant	Well-managed and visible exception pathways (Sindall 2020)
Complete	Lifecycle use of API is well thought through and provisioned (Sindall 2023).
Loose/Low Coupling	Interoperability is achieved without unnecessary knowledge of the operation of the service application/system.
High Cohesion	An API only requires and returns data/classes required to perform the requested action, reducing complexity and context.

3 Cybersecurity

The Department is evolving toward DevSecOps (DevSecOps Reference Architecture 2023), which combines DevOps and integrates cybersecurity principles with an added emphasis on security at each of the DevOps life cycle phases (Figure 3-1).



Source: (DevSecOps Reference Architecture 2023)

Figure 3-1. DevSecOps Infinity Diagram

ZT is a security framework that challenges the traditional approach of trusting entities within a network by default. Instead, it assumes that no entity, whether a user, device, or application, should be trusted automatically. ZT adopts a "never trust, always verify" approach, in which every access request is thoroughly authenticated, authorized, and continuously monitored, regardless of the entity's location or network environment (DevSecOps Reference Architecture 2023).

All APIs need to satisfy and support the requirements of the intended authority to operate (ATO) and align with the intended ZT security posture of the DevSecOps automated pipeline, its automated tests, analysis activities and security posture.

DevSecOps supports the implementation of ZT principles by:

- Incorporating strong identity and access management practices to ensure that only authorized entities have access to resources.
- Implementing granular micro-segmentation to enforce strict access controls and prevent lateral movement within the network.

- Leveraging automation and continuous security practices to continuously monitor and enforce security policies.
- Integrating data-centric security measures, such as data labeling and encryption, to protect sensitive information.

By integrating security practices into the DevOps process, DevSecOps helps organizations build and maintain a strong security posture, which aligns with the principles of ZT (NIST SP 800-207).

3.1 Importance of APIs in Modern Warfare and Emerging Technologies

The increased interconnectivity amplifies the attack surface, however, and introduces new cybersecurity challenges that can potentially compromise mission-critical systems and operations. APIs are susceptible to various attacks, including injection attacks, authentication and authorization issues, availability, and data breaches. Consequently, securing APIs has become an urgent priority for the DoD and other organizations that depend on them. Following are best practices for defending against API threats in the context of emerging technology trends and the evolving battlespace.

3.2 API Cybersecurity Challenges

The Department faces numerous API cybersecurity challenges as it seeks to maintain and modernize its systems. APIs play a crucial role in facilitating communications across a wide range of vital systems, which rely on the secure and timely exchange of information between various Components in support of mission objectives. Cybersecurity challenges include:

- Ensuring the confidentiality, integrity, and availability of API communications and maintaining effectiveness of resilient systems.
- Securing and enhancing data exchange and interoperability capabilities for legacy systems using open standards-based APIs.

Addressing these challenges requires a comprehensive approach involving the development, deployment, and ongoing management of APIs. This approach includes implementing secure coding practices, conducting regular security testing, and continuously monitoring API activity for potential threats. By prioritizing API security, the DoD can ensure the continued effectiveness of its systems while safeguarding sensitive information from adversaries.

APIs are a critical component of modern software systems, enabling communication and data exchange between different applications and services; however, APIs also can be a source of vulnerabilities and threats if they are not properly secured. In the context of the DoD, APIs can

be particularly vulnerable to attacks given the sensitive nature of the data and systems they are used to access.

To mitigate API vulnerabilities and threats, it is important to implement strong API security measures, such as authentication and authorization mechanisms, encryption of data in transit and at rest, input validation to prevent injection attacks, and continuous monitoring of API traffic and logs. In addition, programs should stay up to date with the latest security best practices and regularly review and update API security measures to ensure they remain effective. See also Appendix B: Common API Vulnerabilities and Threats.

NIST provides an API security Special Publication (SP) 800-204, "Security Strategies for Microservices-based Application Systems." NIST provides additional in-depth coverage of the topic in supplemental SPs specific for API security in SP 800-204A, 800-204B, and 800-204C. Although the document is titled for microservices, the "microservices are packaged as APIs" (SP 800-204, section 4.1) to support complex data tasks. The SP specifically addresses API security threats and associated mitigations. Specific threats to APIs include most attacks that normal applications experience including "injection, encoding and serialization attacks, cross-site scripting (XSS), Cross-Site Request Forgery (CSRF), and HTTP verb tampering" (NIST SP 800-204 2019). See also Appendix C: API Security Challenges.

3.3 Cybersecurity Best Practices

API cybersecurity should be designed using ZT principles (NIST SP 800-204), secure coding practices, and defensive measures to protect against attack techniques associated with API environments. Following the best practices in this section can help programs address the challenges DoD faces in securing its systems and data, resulting in more resilient and secure critical systems and operational data against potential threats.

Programs should consider the following six best practices when planning to implement an API project.

3.3.1 Implement Robust Authentication and Authorization Mechanisms

Authentication and authorization (A&A) in an API and data service environment is expected to require internal A&A centralized architecture because of the sheer number of APIs and services interacting. Associated dependencies are realized from using loosely coupled and smaller application bases. The following security controls are recommended:

1. Robust authorization services to ensure availability and timeliness for access decisions.
2. Sensitive data will not use traditional embedded API keys in requests.

3. Digitally signed authentication tokens.
4. If an API key is used, restrictions on where the API and applications may be used.
5. A&A token expiration times as short as possible.
6. A&A token keys produced dynamically in the API and service process using variables represented from the exchange environment.
7. Integration with ZT, for single use or very short time to live (TTL) tokens.

3.3.2 Ensure Input Validation and Output Encoding

Input validation and output encoding are crucial for secure data exchange. They provide a foundation for mitigating API cybersecurity threats, especially within the ZT framework and the specific security challenges the Department faces. Input validation involves verifying and sanitizing user input to prevent malicious code injection and to ensure that only valid and expected data is processed. Output encoding, on the other hand, converts untrusted data into a safe format to prevent XSS attacks and the execution of malicious code in users' browsers. By implementing robust input validation and output encoding practices, the DoD can enhance the security of its APIs, protect sensitive data, and reduce the risk of unauthorized access or manipulation. These measures align with the principles of ZT, which emphasizes continuous verification and strict access controls to mitigate the impact of potential security breaches.

3.3.3 Encrypt and Protect Data in Classified Environment

Encryption and data protection play a critical role in mitigating API cybersecurity threats, particularly in a highly classified environment within the DoD and in alignment with the ZT framework. In a highly classified environment, sensitive information must be securely transmitted and stored to prevent unauthorized access or disclosure. Encryption ensures that data is transformed into an unreadable format, making it inaccessible to unauthorized individuals. By implementing strong encryption algorithms and secure key management practices, the DoD can protect the confidentiality and integrity of its data, even in the event of a breach. This protection aligns with the principles of ZT, which emphasizes the need to verify and protect data at all stages of its life cycle, regardless of the network or environment in which it resides.

3.3.4 Monitor and Log for Early Threat Detection and Response

Monitoring and logging are essential components for early threat detection and response in the context of API cybersecurity threats. By implementing robust monitoring and logging practices, the DoD can continuously monitor its APIs and systems for suspicious activities, unauthorized access attempts, and potential security breaches. This monitoring enables the timely detection of threats and facilitates a rapid response to mitigate the impact of any potential breaches. In

addition, comprehensive logging allows for detailed analysis and investigation of security incidents, aiding in the identification of vulnerabilities and the implementation of necessary remediation measures. These monitoring and logging practices align with the principles of ZT.

3.3.5 Tailor API Gateway and Firewall Protection to DoD Requirements

Since the API gateway is the primary component to effective API implementation, management, and security, it should be equipped with infrastructure services appropriate to mitigate the growing list of threats. At the least, these enterprise services should include service discovery, authentication and access control, load balancing, caching, application aware health checking, attack detection, security logging and monitoring, and circuit breakers (NIST SP 800-204).

Specific security strategies for API gateway (GW) include (NIST SP 800-204):

1. Integrate an identity management application to provision credentials before API activation.
2. API GW should have a connector to generate an access token for a client request.
3. Securely (HTTPS, SSH, OpenSSH, Type1 NSA) channel all traffic to a monitoring and analytics solution for attack detection and performance degradation.
4. Ensure distributed API GW deployments and microgateways (GW customized and closer to API and service) include a token exchange service between GWs. Edge GWs should have tokens with broader permissions, and internal microgateways should have more narrowly defined permissions, enabling a least privilege paradigm.

3.3.5.3 Securing API Service Discovery

Securing API service discovery is crucial for maintaining the availability and proper location of services as needed. This importance is particularly vital in the virtual and cloud environments where they might need to be replicated and relocated for various reasons, including security considerations. It is essential that service discovery not only facilitate the connection between the clients and services but also ensure the authenticity and integrity of the services provided.

Loose coupling should be used, and APIs should not include self-registration and deregistration capability. If an API and associated service crash or they are unable to handle requests, the inability to perform deregistration affects the integrity of the data and information-sharing process. Reliance on API local cache data should be used only if dependent servers are unavailable.

3.3.5.4 Implementing Circuit Breakers in API and Services to Prevent Cascading Failures

Circuit breakers are a common technical implementation for minimizing cascading failures. Circuit breakers prevent delivery to an API and associated service that is failing. This technique helps prevent security attacks such as denial of service and brute force attempts (i.e., login attempts, erroneous inputs, and code injects). Three options for deploying circuit breakers are client-side, server-side, and proxy implementation. Proxy implementation is recommended because it avoids placing trust on clients and APIs.

3.3.5.5 Data Integrity Assurance

Data integrity assurance is the assurance that digital information is uncorrupted and can be accessed or modified only by those authorized to do so (e.g., system database administrator). There are four types of data integrity to consider: entity, referential, domain, and user-defined. Data integrity assurance is a critical consideration for APIs and their underlying data.

3.3.5.6 Other API Security and Release Recommendations

With new APIs, versions, and associated services (Corbo 2023):

1. Canary release tactic should be employed, thus limiting the number of requests and use of a new API. This is to protect failure and erratic behaviors of both clients and API as use cases may not meet the expectations despite extensive testing.
2. Usage monitoring of existing and new version should steadily increase traffic to new version.

3.3.5.7 Session Persistence

1. Session information needs to be stored securely.
2. Internal API and service authorization tokens must not be provided to the user.
3. User session tokens must not be provided beyond GW used for policy decisions.

3.3.6 Ensure API Security Testing and Compliance in the DoD

By conducting API security testing, the DoD can proactively identify and address potential security risks, ensure compliance with security standards and regulations, and enhance the overall security posture of its systems and data. These testing practices are crucial for maintaining the integrity, confidentiality, and availability of APIs within the DoD environment.

Static application security testing (SAST) and dynamic application security testing (DAST) should be conducted continuously as part of the continuous integration/continuous deployment (CI/CD) pipelines and other manual testing (e.g., penetration) should be conducted prior to initial

deployment and at regular intervals (e.g., monthly, quarterly, yearly), based on program security needs.

3.3.6.3 Types of API Security Testing Relevant to the DoD

Several types of testing are relevant for DoD APIs, including (1) functional testing, which ensures APIs perform as intended and handle inputs correctly; (2) penetration testing, which simulates real-world attacks to identify vulnerabilities and weaknesses; (3) fuzz testing, which involves sending unexpected or malformed data to an API to uncover potential security flaws; and (4) security code reviews, which involve analyzing API source code to identify security vulnerabilities.

3.3.6.4 Common Tools and Techniques for API Security Testing in a Defense Context

Tools and techniques to ensure compliance in API security testing include OWASP ZAP, Burp Suite, JMeter, Postman, and Nessus, which help identify vulnerabilities and weaknesses in APIs. These tools and techniques allow DoD to proactively identify and address potential security risks, ensure compliance with security standards and regulations, and enhance the overall security posture of systems and data.

3.3.6.5 API Security Testing and Compliance within the DoD

DoD can follow several best practices to improve API security testing and compliance. Functional testing, penetration testing, fuzz testing, and security code are essential to identify vulnerabilities and weaknesses in APIs. Regularly updating and patching APIs, as well as monitoring and logging API activities, can help a program detect and respond to potential security incidents. Adhering to industry standards and regulations, such as those outlined by the DoD and NIST, is essential for maintaining compliance.

4 Design and Implementation Principles

To ensure the success of an API project, an "API-first" strategy emphasizes the key principles of modularity, scalability, and reusability. By adopting this strategy, a program places priority on the design and development of the API before implementing the underlying system. Creating a modular architecture enables scaling, reuse across various applications and platforms, and seamless integration, flexibility, and enhancements. This approach also encourages a more efficient and collaborative development process, allowing teams to work in parallel, with front-end and back-end developers focusing on the team's areas of expertise. Ultimately, an API-first strategy sets the stage for a robust and adaptable system that can meet the evolving organizational and stakeholder needs. The following paragraphs discuss seven design and implementation principles that support an API-first strategy.

4.1 Common Data Model

It is essential to establish a Common Data Model (CDM) early in the design and implementation phase that can be used across all API endpoints. This CDM serves as a standardized schema or structure for organizing and sharing data, ensuring consistency and interoperability between different API components and applications. Defining a common data model creates an understanding of how data should be shaped and shared, enabling rapid unification of data and facilitating seamless integration between various systems and services. This promotes data consistency, reduces complexity, and enhances data interoperability, allowing different applications to communicate effectively and exchange information seamlessly.

Implementing a CDM not only streamlines the development process but also improves data quality, accuracy, and reliability, as it eliminates the need for data mapping and transformation between different systems. In addition, the CDM enables easier data integration with industry-wide standards and frameworks, facilitating collaboration and data exchange with external partners and stakeholders. Early and comprehensive adoption of a common data model establishes a solid foundation for the API ecosystem, ensuring consistency and interoperability throughout the entire system.

4.2 Open Standards and Protocols

In the design and implementation phase, the API project should leverage open standards and protocols to ensure compatibility and interoperability with other systems and applications. Adoption of widely accepted standards such as Representational State Transfer (REST), Extensible Markup Language (XML), and JavaScript Object Notation (JSON) helps enable seamless communication and data exchange between different components of the API ecosystem.

REST provides a lightweight and scalable architectural style for designing networked applications, while JSON offers a simple, compact, and human-readable format for representing data. These open standards promote flexibility, allowing developers to easily integrate with the API, reducing the learning curve associated with understanding proprietary protocols.

In addition, incorporating Open Authorization (OAuth) as a security protocol enhances the security and trustworthiness of the API. OAuth enables secure and delegated access to protected resources by providing a standardized framework for authentication and authorization. OAuth ensures that only authorized users or applications can access sensitive data or perform specific actions within the API. This protection not only enhances the security of the API but also simplifies the integration process for developers, as they can leverage existing OAuth libraries and frameworks to handle authentication and authorization (OAuth 2.0 2023).

Embracing open standards and protocols will lead to an API ecosystem that is compatible with a wide range of systems and applications. This approach promotes interoperability, allowing the API to seamlessly integrate with existing infrastructure and enabling developers to leverage existing knowledge and tools. The use of open standards reduces vendor lock-in and fosters a collaborative development environment, as developers can easily understand and work with the API. Overall, adopting open standards and protocols is a key aspect of designing and implementing an API that is accessible, interoperable, and developer friendly.

Given the DoD's global operation environment, across all time zones, one important standard to consider is the use of the ISO-8601 (2019) date and time format, which supports a variety of flexible use cases and formatting options. ISO-8601 allows local date and time to be expressed using a time zone designator, with extreme precision. ISO-8601 is critical in supporting interoperability among theater, combatant commands, and continental U.S. C2 environments.

4.3 Design for Security Compliance

For the DoD, designing the API with security compliance in mind is critical. Incorporating security measures from the outset protects the API from unauthorized access and potential data breaches.

One of the main security protocols to consider is Secure Sockets Layer/Transport Layer Security (SSL/TLS), which provides encryption and secure communication between clients and servers. Implementing SSL/TLS ensures that data transmitted between the API and its consumers is encrypted, preventing eavesdropping and unauthorized interception of sensitive information.

In addition, using industry-standard authentication and authorization protocols such as OAuth and JWT (JSON Web Tokens) enhances the security of the API. OAuth enables secure and delegated access to protected resources, ensuring that only authorized users or applications can

access specific endpoints or perform certain actions. JWT provides a secure method for transmitting claims or assertions between parties, allowing for secure authentication and authorization.

API projects should design the API with security compliance in mind to establish a robust security framework that protects sensitive data and prevents unauthorized access. This framework not only safeguards the integrity and confidentiality of the API but also helps to satisfy industry and regulatory compliance requirements. It is important to stay up to date with the latest security best practices. Projects should regularly review and update security measures and address emerging threats and vulnerabilities.

Overall, incorporating industry-standard security protocols such as SSL/TLS, OAuth, and JWT into the API design ensures that security is a fundamental aspect of the API ecosystem, protecting data and the privacy of users (e.g., soldier personally identifiable information (PII), medical systems protected health information (PHI)).

4.4 Developmental Testing and Validation Processes

Establishing a robust testing and validation process to ensure the quality and reliability of the APIs is also essential. By implementing comprehensive testing methodologies including unit, integration, and section tests, issues or vulnerabilities can be identified and addressed before they reach the production environment. Three common testing concepts in a robust process are unit testing, integration testing, and end-to-end testing.

Unit testing involves testing individual components or units of code to ensure they function correctly in isolation. Unit tests help identify and fix bugs, validate the behavior of individual functions or modules, and provide a solid foundation for building more complex features.

Integration testing involves testing the interaction between different components or modules within the API to ensure they work together seamlessly. Integration tests help identify any issues that may arise when different parts of the API are combined, ensuring the overall functionality and reliability of the system. Integration testing between components and the APIs that support them is also necessary to validate that the APIs themselves are functioning correctly.

End-to-end testing validates the entire flow of the API, simulating real-world scenarios and user interactions. This type of testing ensures that all components, integrations, and dependencies work together as expected, providing a holistic view of the API’s performance and functionality.

By implementing a comprehensive testing and validation process, any issues or bugs can be identified and addressed early in the development cycle, reducing the risk of encountering

problems in the production environment. This helps ensure that the API functions as intended, delivers the expected results, and provides a positive user experience.

API projects should automate testing processes as much as possible, using tools and frameworks that facilitate test automation. Automation allows for faster and more efficient testing, enabling CI/CD practices. In addition, incorporating security testing, such as penetration testing and vulnerability scanning, helps identify and mitigate potential security risks.

A robust testing and validation process is crucial for ensuring the quality, reliability, and security of an API. By conducting thorough unit, integration, and end-to-end testing, any issues can be identified and addressed early on, leading to a more stable and reliable API.

Data used to test APIs as well as results should be submitted to a centralized database such as cloud hybrid edge-to-enterprise evaluation and test analysis suite (CHEETAS) for testing and evaluating data from test range events or other.

4.5 Collaboration and Communication

Collaboration and communication among developers, architects, and other stakeholders are essential elements in the successful development and implementation of APIs. A collaborative culture with fluid communication paths and consistent internal feedback loops helps ensure that the API will meet the needs and expectations of all parties involved. In addition, this communication helps establish a culture of psychological safety (McKinsey & Company 2023) and professionalism that focuses on respect for all team members.

API development also requires close collaboration between consumers and producers. Consumers need to stay up to date on the latest changes to how the API works, while producers need feedback from consumers to ensure they are building the right thing. A communication plan facilitates frequent and iterative information flow to the consumers, provides consumer feedback, provides recommendations, and allows reporting on issues and bugs. In addition, holding weekly scrums or monthly “ask me anything” engagements further enhance the relationship. Lastly, fostering a culture of collaboration helps create a robust feedback cycle that allows producers to better understand consumer needs and supports continuous improvement and iteration.

To facilitate collaboration and communication, teams can use tools such as API documentation, developer portals, ChatOps, collaboration software, and forums. API documentation serves as a comprehensive resource that provides information on how to use the API, its endpoints, parameters, and response formats. Developer portals act as a centralized hub where developers can access documentation, explore API features, and engage with the API community. Chatops typically provides a highly interconnected environment across topic threads to obtain immediate answers from appropriate team members. Collaboration software typically provides an

ecosystem of online tasking, communications and meeting capabilities. Forums provide a platform for developers to ask questions, share insights, and provide feedback on the API. Projects should consider experimentation and pilots to establish which tools are most effective.

API projects should use these and other effective tools to create an environment that encourages open communication and collaboration. This open environment enables API developers, architects, and other stakeholders to share ideas, address concerns, and work together toward building a robust and user-friendly API.

Overall, encouraging collaboration and communication is crucial for the success of an API project. Fostering a collaborative environment and using tools facilitates effective communication, enhances feedback for all parties involved, and helps the project deliver a seamless experience for API consumers.

4.6 API Parameters for Pagination, Sorting, and Filtering

Pagination is essential when dealing with large result sets. By using parameters such as “limit” and “offset,” clients can control the number of items returned per page and navigate through the result set. This approach prevents overwhelming the client with a massive amount of data and improves performance by reducing the payload size.

Sorting is another important capability that can be achieved through API parameters. Clients can specify the sorting criteria using parameters like “sort” and “order.” This allows them to retrieve data in a specific order, such as ascending or descending based on a particular field. Sorting empowers clients to organize and analyze the data according to their requirements.

Use of filtering is critical for harnessing the power of APIs. Filtering allows retrieval of data based on specific criteria (e.g., datetime, domain, category, location) and reduces or eliminates the amount of irrelevant information retrieved. For larger data sets, filtering can significantly improve performance and efficiency of request responses, post-result processing, network communications, and system utilization.

API parameters provide flexibility and customization options to clients, allowing them to tailor API responses to their requirements. By supporting pagination, sorting, and filtering capabilities through parameters, APIs can deliver a more efficient and personalized experience to clients. It is important for API developers to design and document these parameters effectively, ensuring that clients understand how to use them correctly and take full advantage of the API’s capabilities.

Overall, API parameters are a powerful tool for enhancing the usability and efficiency of REST APIs. They enable clients to control the data they receive, navigate through large result sets, sort data according to their needs, and filter out irrelevant information. By incorporating these

capabilities into API design, developers can provide a more flexible and user-friendly experience for API consumers.

4.7 API Metrics

Several metrics should be considered to measure API performance and effectiveness. Metrics help assess the performance, availability, and usage of APIs. Some API-specific metrics include:

1. Response Time: Measures the time it takes for an API to respond to a request. This metric helps understand the performance of the API and identify any bottlenecks or latency issues.
2. Error Rate: Tracks the percentage of API requests that result in errors. This metric helps identify any issues with the API's functionality or stability.
3. Availability: Monitors the uptime and availability of the API. This metric helps ensure that the API is accessible to users and identifies any downtime or service interruptions.
4. Usage and Traffic: Measures the number of requests and the volume of data being processed by the API. This metric helps understand the usage patterns and scalability requirements of the API.
5. Latency: Measures the time it takes for data to travel from the client to the API server and back. This metric helps identify any network or infrastructure-related issues that may impact API performance.
6. Rate Limiting: Monitors the number of requests per second or minute to enforce rate limits and prevent abuse or overload of the API.
7. Authentication and Authorization: Tracks the success rate of authentication and authorization processes for the API. This metric helps ensure the security and integrity of the API.
8. SLA Compliance: Measures the compliance of the API with the defined Service-Level Agreements (SLAs). This metric helps assess whether the API is meeting the performance and availability targets set for it.

These metrics will provide valuable insights into the performance, usage, and reliability of the APIs. Projects should regularly monitor and analyze these metrics to identify areas for improvement and to make data-driven decisions that optimize the API's performance and improve user experience.

It is important to define specific goals and thresholds for each metric based on system requirements and consumer expectations.

5 Development, Security, and Operations (DevSecOps)

There are several key documents to consider when planning and implementing DevSecOps:

1. The DoD Enterprise DevSecOps guidance (DevSecOps Reference Architecture 2023) is the Department's primary guidance to develop and deliver secure software using modern practices. The guidance outlines principles including API security testing that aligns with industry best practices and supports the DevSecOps culture.
2. NIST SP 800-204 (e.g., Microservices) (2019), NIST SP 800-207 (e.g., ZT) (2020) and other industry standards also help ensure APIs meet operational security requirements.
3. DoD Instruction 5000.87, "Operation of the Software Acquisition Pathway" (2020), defines DevSecOps as, "An organizational software engineering culture and practice that aims at unifying software development, security, and operations." This instruction establishes policy for the software acquisition pathways and calls out DevSecOps as a recommended development strategy.
4. The Carnegie Mellon University Software Engineering Institute (SEI) described DevSecOps as "an entire socio-technical environment that encompasses the people in certain roles, the processes that they are fulfilling, and the technology used to provide a capability that results in a relevant product or service being provided to meet a need" (SEI 2022a).

The processes and practices of DevSecOps are not significantly affected by whether the software in question is an API or not; however, APIs necessitate additional verification prior to release.

Implementation of a versioning strategy including backward compatibility, which will affect the DevSecOps approach or automated pipeline flow and processes. Backward compatibility between versions creates challenges for integrating and releasing new code; however, strategies exist to make this backward compatibility possible: releasing new APIs on unique resource locators (paths or ids), feature flags, content negotiation headers, creating optional fields on API requests or even maintaining multiple versions of APIs deployed at the same time. Coordination with consumers or clients may be needed to ensure everyone is aware of a deployment with a breaking change. If the team keeps their API always backward compatible, it can be released once all tests and acceptance criteria pass. How an API is versioned and developed will affect the deployment strategy, and the trade-offs must be considered.

Similarly, the architecture of an API will affect its deployment. If an API is served by a single physical machine, it may be simple to re-deploy a new version, but considerations about downtime and user experience then need to be considered. Conversely, if the API is designed in such a way that user requests are load balanced across several machines or a cluster, a team may

be able to perform a rolling release that deploys to one machine at a time. Again, this is intertwined with backward and forward compatibility of the API. In these ways, the use of APIs in a system will affect the DevSecOps pipeline and process.

The DoD Chief Information Officer (CIO) provides general documents on DevSecOps at https://dodcio.defense.gov/library/.

5.1 DevSecOps Enabling Technologies

This section addresses five key DevSecOps-related technologies: version control, secrets management, policy enforcement, embedded security, and the CI/CD pipeline flow.

In the DevSecOps Platform Independent Model (PIM) (SEI 2022b), a requirement of maturity level 1 (of 4) for a DevSecOps program is to have a source code repository and to maintain version control of that source code (these are requirements Dev_7 and Dev_7.1 in the model). This use of version control is essential to enabling a DevSecOps project. Version control allows for consistent change management and integration of new changes into the deployed version of the API. Understanding what files, versions, features, and defects are present in a version of an API is necessary for each step in the DevSecOps process (DoD CIO Fundamentals Guidebook March 2021). In addition, being able to publish a new version of an API or roll back to a previous version is dependent on a version control strategy.

Secrets management is particularly important for producers and consumers of an API. It is common for an API to require authenticated and authorized access to call certain endpoints or certain functions. Developers of APIs should use well-known industry standards such as Multi-Factor Authentication and OAuth 2.0 for delegated authentication. Authorization services (e.g., Policy Enforcement Point (PEP), Policy Decision Point (PDP)) should be kept separate from the API and should compute decisions based on user roles, scopes, and policies. These should be based on the principle of least privilege, giving clients only the minimum necessary permissions.

Once authenticated and authorized, an API consumer will need to securely store and rotate credentials, secrets, and encryption keys, which can be commonly achieved through a secrets management provider. Examples of providers include Amazon Web Services (AWS) Secrets Manager, Azure Key Vault, and Hashicorp Vault. In addition, detailed auditing and logging of authorization failures should be implemented in a centralized service away from the API that does not collect any PII. A well-designed authorization mechanism with a centralized policy store is crucial for securing APIs and protecting data from unauthorized access.

The API is generally the most exposed portion of a code base or system to which users and malicious actors may have access. Baking in quality and security to the code that implements an API is crucial to both security of the system overall and to meeting requirements. For an API in a

weapon system, these are typically defined in a Contract Data Requirements List (CDRL), which references a Data Item Description (DID). The DID defines data format and content requirements or another interface document. Web-based or administrative APIs may provide requirements and data definitions in a different format.

In either case, a pipeline can ensure the DIDs or data format requirements are being met by the system as implemented. A team then can answer specific questions about the APIs such as, "Am I exposing any vulnerabilities through poor code conventions? Do the functions handle the nominal cases and edge cases as specified? Am I exposing any secret keys or tokens in the built software?" All these aspects are especially important for API development as the API is the user's interface to the code (be it a human or non-person entity). Tools exist that combine these ideas as well as create documentation of user acceptance testing in a matrixed organization.

As discussed in the testing section, there are many recommended API test and evaluation roles, and having a robust CI/CD pipeline can free up developers' time to fill these roles. In addition, pipelines can provide feedback on the progress, conformance, and compliance of software development and developed products. This can be in the form of "gates checks" on the status of test results, manual approvals, or other criteria before a new version is published. For APIs, these checks could include whether a new version has been approved by a dedicated test person.

5.2 Monitoring and Logging

APIs provide a unique opportunity to measure and monitor a system. This measuring and monitoring plays into collecting information during the operations cycle of DevSecOps as well as enabling ZT through monitoring and logging of users. In addition to enabling security through the feedback they provide, APIs can enable a better user experience if the logging or monitoring is able to determine common usage errors.

When considering APIs in a DevSecOps context, APIs can enable operations monitoring. According to the DoD DevSecOps reference architecture (DoD CIO 2019) operations monitoring should be automated and APIs provide a means for this automation. The metrics will be different for each system.

Monitoring API usage can help inform a project's versioning strategy as identified in Section 2 (API Governance). If a breaking change is to be made, some users may have trouble upgrading and may need to stay on an older version of the API. If the project is willing to support multiple versions, monitoring can help inform when all or most users have switched to a new version and support for an old version can be dropped. In the context of deployed systems, this monitoring of multiple concurrent versions is crucial to make sure users in the field are not cut off.

APIs also help in monitoring a system by providing transparency of usage. Users that query a monitored API can provide valuable information for analysts, such as identifying user interaction patterns not obvious in the previous development cycles, detecting errors and incorrect usage, etc. In addition, as discussed in the Section 6 (Testing), monitoring and rate limiting (or an API Gateway that combines all these components together) can help prevent a denial-of-service attack. This protection provides a better experience to legitimate users.

When considering APIs for monitoring in a ZT context it is necessary to aggregate log data for building patterns of usage for consumers of the API. According to the Cybersecurity and Infrastructure Security Agency (CISA), an optimal ZT will make authorization decisions based on "incorporating real-time risk analytics and factors such as behavior or usage patterns" (CISA 2023, 23). To be able to identify and check against legitimate usage patterns, however, one must have a log of data sufficiently large to prove robust for analysis.

The API Gateway is an architectural pattern that enables monitoring. According to the F5 Glossary, an API Gateway is "a component of the app-delivery infrastructure that sits between clients and services and provides centralized handling of API communication between them" (F5 Glossary n.d.). This pattern allows for an intermediary between clients and the API implementations to monitor and redirect traffic as needed and to provide discoverability, authentication, and authorization capabilities. In the case of a denial-of-service attack (or even an unintentional system overload), an API gateway can be configured to drop traffic from some clients.

DevSecOps and Agile software development are based on the ability to receive feedback from the users and incorporate it into the project plan to continuously improve. The ability to collect metrics and information from an API can make them an asset to this end. While API development is largely the same as for any other software, the security and operations of an API may need to be tailored to meet requirements of different architectures and operational environments.

6 Testing

If builders built buildings the way programmers wrote programs, then the first woodpecker that came along would destroy civilization.

—Gerald Weinberg, former manager of operating systems development for the NASA Mercury Program

APIs, like any software, may have bugs, which is anything deviating from the expected functioning of the system. Vulnerabilities are a subset of bugs through which the system can be exploited (intentionally or unintentionally) to create significant risk for the operation of the system, people, and organization. What constitutes "risk" depends on the system component, data, personnel, and other factors (Mead and Woody 2017). APIs are particularly risky because they serve as system access points, so a bug or vulnerability in an API can enable bad actors to enter the system and cause it to behave unexpectedly. Preventing this from occurring requires a good design, implementation, and continuous testing of the software product to provide high assurance the system functions only as intended.

Testing is an integral part of the software development life cycle assuring the software product is working as expected and finds potential sources of risk from system vulnerabilities. Categories of testing include not just software but documentation, personnel, system integration, system-of-systems, and more (Firesmith 2015). This section will focus on software testing methods specific to APIs based on available data and best practices.

6.1 Software Testing

Software testing comes in two basic varieties: *manual* and *automated*. Manual testing provides qualitative feedback from the perspective of the user, whereas automated testing is useful for getting fast and repeatable feedback. Using automated tests has been demonstrated to significantly improve software quality and reduce manual labor in the long run, even though there is an upfront cost in creating the tests (Kumar and Mishra 2016) (Williams 2009). Some of the most common types of automated tests are unit testing, integration testing, and application testing (sometimes called *end-to-end [E2E] testing*). Below is a brief description of each type of testing and how it is used:

6.1.1 Unit Testing

Unit testing is low-level testing focusing on a small part of the software system usually written by developers using a testing framework[1]. Unit tests can be written for both the interface and

[1] A testing framework is a combination of practices and tools designed to help test more efficiently, usually in the form of libraries that assert code behavior.

implementation of an API. Unit tests can be documented early in the design phase, added during development as developers discover new data paths, and added long after implementation to provide regression testing for any kind of refactor. Because it is infeasible to test all the possible combinations of inputs an API can have, developers can use two main strategies of unit testing for dividing the input space: deterministic and non-deterministic.

The deterministic strategy uses equivalence partitioning and boundary value analysis (Humble and Farley 2015). Equivalence partitioning divides a set of test conditions into different testable segments. Testing discrete segments provides the benefit of dramatically reducing the range of inputs to test while keeping the testing values relevant to what the software is supposed to do.

For example, the grades for an exam may be A (90-100), B (80-89), C (70-79), D (60-69), and F (0-59); each grade can be thought of as a different segment to be tested. In API unit testing, rather than testing the input range from 0-100, a project can test a single value from each segment (e.g., 5, 63, 78, 85, and 92). A testing segments approach reduces the range of inputs to test by providing a representative sample. Boundary value analysis tests the values between the partitions. So, in the case of the exams, unit tests would cover input values 0, 60, 90, and 100, which are at the edge. Combining these techniques provides a way to thoroughly test a variety of values without having to exhaustively test the entire input space.

The non-deterministic strategy uses a fuzz tester (OWASP 2024), an automated, non-deterministic type of testing tool that excels at generating a range of testable inputs in a short amount of time. Fuzzers vary in how structured their inputs are and how they change or mutate over time. Fuzzers also can vary from black-box (not considering the internals of the program) to grey-box (having some idea of the internal structure and the kinds of inputs it accepts). To increase the coverage of inputs and outputs in an API, a fuzzer should be combined with automated unit tests. There are several open-source fuzzing tools, such as RESTler (GitHub 2024) and Dredd (2024), which will consume an OpenAPI specification and automatically generate and execute test cases against an implementation.

6.1.2 Integration Testing

Integration testing is used to test whether multiple units of code developed separately (such as services, modules, or projects) work correctly when connected, or integrated, with each other. For example, an integration test for an API that relies on a database connection will provide a mock or test double[2] for the database and ensure that it functions correctly during its expected

[2] In this context a mock or test double is narrowly targeted to replicate the behavior of real software as it pertains to the test in a controlled way. This is short of a digital twin which should replicate a real software or system in a general way.

(or unexpected) behavior. Integration tests assume that the stubbed services and dependencies are accurate and faithful representations of the real services.[3]

API integration tests should cover the issues commonly encountered in distributed architectures and systems (i.e., concurrency, fault tolerance, failure recovery) (Anderson 2020, 243). Integration testing also would include APIs that need to function in a denied, disrupted, intermittent, and limited (DDIL) environment. In some cases, these adverse environments require alternate code paths or, in the case of an API, a different implementation to maintain operation. Even when co-located with other software on the same physical system, components that interact through APIs can fail when multiple consumers try to use the API simultaneously. Orchestrating integration tests that target these cases can be important for systems (e.g., a database API) that need to be atomicity, consistency, isolation, and durability (ACID) compliant.

Fault tolerance and failure recovery are closely related. Tests for an API should ensure that an error-handling mechanism is exercised when the API processes bad data or is in an error state. Failure recovery is then activated to perform the steps to move past the error state and provide service to consumers. In some systems, failure recovery can be entirely manual or infeasible. Even in these cases, ensuring the API sends the correct signals or returns the right error code can be important. Tests that cover both fault tolerance and failure recovery provide robustness and availability to an API.

Two ways to test for fault tolerance are *stress testing* and *graceful degradation testing*. An API will go through a pipeline stage where different load parameters (# of threads, connections, etc.) are increased and decreased past the API's expected operating range to test its resilience. Stress testing also helps uncover potential vulnerabilities the system exposes when services are strained or taken past their expected working range by, for example, overfilling buffers and writing in over memory, or unexpectedly dropping transactions if the number of connections it can handle are exceeded. In graceful degradation testing, the system responds gracefully or in a "least surprising" way to its end users if the service cannot fulfill requests.

Chaos engineering is a more advanced version of graceful degradation testing in which faults are randomly injected into the system to test its resilience and error-handling capabilities.

6.1.3 Application Testing

Application testing verifies that the developed application in a representative environment meets the stated requirements. This type of testing examines the input and output of the API, including the other services it relies on. This is *live* testing, and it usually involves using *test data* in a *test*

[3] These services may change over time unexpectedly, in which case a Contract Test (see section 6.4) will provide a valuable stopgap for the developer to ensure that mocks map to the real service's interface and behavior.

database in a *testing environment* (see Section 6.2). In this manner, application testing also can be considered *user acceptance testing*. Ideally, application testing is conducted during one of the later stages of the CI/CD pipeline, running a series of automated tests on a testing environment and assessing that both the *happy path* (i.e., expected behavior) and a few *bad paths* (i.e., unexpected behavior) are tested. Automating this step eliminates a potential manual checkpoint to enable changes to be deployed quickly. Certain industries require a human to review changes before deployment, but this review can be performed after application testing and as the final step before deployment.

When considering application testing for APIs, it is important to consider adversarial users. This is also called threat-based cyber testing. Adversarial users can exploit an API in several ways. In addition to the previously mentioned issues with authentication, the Open Worldwide Application Security Project (OWASP) describes "unrestricted resource consumption" and "unrestricted access to sensitive business flows" as two of the top 10 risks for APIs. Depending on the vector, testing for these vulnerabilities may be performed as an integration test or an application test. Ultimately, ensuring all services react appropriately in the development environment is essential to catch errors induced by bad actors in production.

6.1.4 Other Types of Testing

Other types of software testing exist, including coverage testing, SAST, and dependency scanning. Each of these tests provides an additional layer of verification by testing different parts of an application under varying conditions. Because software testing is context-dependent and exhaustive testing is not possible, developers and testers should be knowledgeable about the various testing frameworks and their appropriate use to optimize the amount of testing based on the requirements, quality attributes, and overall risk assessment of the API.

An effective API testing strategy, leveraging other tests (e.g., coverage, SAST, and dependency scanning foundation) provides programs with a strategic advantage and establishes a path to a successful deployment and high assurance. As Justin Searls from Test Double points out, "Nearly zero teams write expressive tests that establish clear boundaries, run quickly and reliably, and only fail for useful reasons. Focus on that instead" (Searls 2021).

6.2 Test Environments

When testing APIs, a test environment is required. The *test environment* is the set of systems that tests run on and the resources needed to create conditions as close to production as possible. Things that make up a test environment include hardware configuration of servers, operating system configurations, middleware, and services that support the application (such as databases and authentication systems) and the API system(s). According to the Software Engineering Body

of Knowledge (SWEBoK), a test environment "should facilitate development and control of test cases, as well as logging and recovery of expected results, scripts, and other testing materials" (IEEE Computer Society 2023). The objective is to ensure that testing can be repeatedly and reliably performed such that the results of the tests provide confidence to developers and product owners that the system is ready for launch.

6.2.1 Ephemeral versus Persistent Environments

Two main forms of testing environment are *ephemeral* and *persistent*:

- An *ephemeral environment* is created to execute tests and is removed once tests are complete. It relies on the infrastructure-as-code (IaC) principle. This method of testing can save money and resources in a cloud-based or shared computing environment (as long as provisioning resources is not so expensive to do). Given that resources are used only while tests are executing, hourly or per-minute compute costs are minimized (Villalba 2023). An added benefit of using this environment is the decreased likelihood of configuration drift, which is the tendency of systems to change over time and which results in small software differences from the originally deployed system. Examples of ephemeral environments are short-lived containers, serverless components, and environments built through IaC that are rebuilt and destroyed in each continuous integration (CI) build cycle.
- A *persistent environment* is long-lived and is not decommissioned or spun down at the end of testing. This type of environment offers the benefit of pre-allocated resources to execute the tests, potentially leading to quicker test starts; however, since it is a long-lived infrastructure, continued configuration management becomes an issue that must be handled. This configuration management includes updates to the system and correctly setting the system's initial state. Having a known initial state ensures that tests can be consistently repeated, with the setup process often being the most time-consuming aspect. Ephemeral environments are generally not affected by prior states. Examples of persistent environments are Virtual Machines or servers that are specifically provisioned as "Test" servers.

When developing a plan for the test environment, the focus should include feedback and participation from developers, project managers, and applicable stakeholders. To adequately test the system against performance requirements, the plan should incorporate expertise and input from these participants and be representative of the operational environment. Section 4.3.3 of the Cybersecurity Test and Evaluation Guidebook ((DoD February 2020) provides helpful guiding questions about the who, what, when, where, and how of planning a test environment. The questions include considerations of test infrastructures (e.g., appropriate labs) that can have a

long lead time to set up. In the DevSecOps Playbook (DoD CIO September 2021), the section titled "Play 2: Adopt Infrastructure as Code (IaC)" applies to both persistent and ephemeral test environments. The playbook provides information about the key benefits of IaC. For API testing, IaC has the benefit of providing a known state for starting a test. Many APIs are stateful, so testing requires setting up a known good (or bad) state. Automating the setup of ephemeral test environments is key to enabling developer velocity.

APIs frequently require a network to operate. As per the previous recommendation, the test environment should be as representative as possible, which includes making the networking environment similar to production, while at the same time putting safeguards so external systems are not affected by testing. This testing could include emulating the various network segments (e.g., local area network (LAN), wide area network (WAN), tactical data link (TDL)) to test properties such as reduced throughput, error rate, or high latency. This would be critical for APIs that are required to work in a DDIL context.

Containers[4] are another useful tool for quickly packaging and shipping an API to different environments and can be a crucial part of the testing strategy. Containers isolate processes that provide repeatable, sandboxed environments that can be packaged in a standardized format.

6.2.2 Test Data

The U.S. Department of Defense *Data Strategy* (2020) goals are summarized by VAULTIS in Section 1.4. These goals should be applied to operational APIs, and the data used to test the APIs should be congruent with these goals. Especially, it is important to ensure realism in the environment and in the test data. The test data must have the same format (interoperable) and have real or representative values (understandable) that are available for API developers and consumers to use (accessible) while not leaking any sensitive data, such as PII, PHI data, or other data (secure).

Often, test data may not be readily available. In these cases, it may be necessary to generate simulated test data. Again, this data should be trustworthy, and producers and users of the simulated data should be confident that it represents real-world data (DoD Issues New Data Strategy 2020). Conversely, for some systems such as those for machine learning (ML), environments may require so much test data that configuration management is a concern. Both the DoD Data Strategy (2020) and the Data, Analytics, and Artificial Intelligence Adoption Strategy (June 2023) provide evaluation criteria for data integrity and accessibility. When

[4] Containers are a package of software and all dependencies that virtualize the operating system to allow execution in many different environments with minimal configuration of the host system.

considering AI/ML systems specifically, data gathering, preprocessing, and use come with specific considerations that are out of scope of this guidebook.

6.2.3 Testing Suite Infrastructure

An automated testing suite with slow tests can tempt developers to skip writing additional tests or running them, which defeats the purpose of the testing suite. As such, it is important that a project's test suite be both fast and reliable. What is *fast*? Some practitioners say that tests should take no longer than 10 seconds to run (Seemann 2012) based on research that 10 seconds is the maximum amount of time people will wait while maintaining focus on the task at hand; however, 10 seconds may be too fast for the project's needs or capabilities. If so, team members must agree on how long they are willing to wait without having to switch to another task. Achieving these agreed-on speeds requires configuring the test framework to use parallelization as much as possible.

Other testing strategies for longer running tests include nightly or weekly builds of a software system. These tests are run less frequently but still on a regular schedule. While not a replacement for on-demand testing, this strategy can help offload long-running tests or builds to servers overnight. An example of where this may be applicable is in an end-to-end test of a microservice based API. Each service behind the API has its own pipeline that runs quickly and maybe performs its own unit-level testing. Every night a more extensive pipeline to build the entire product can be run that would perform integration or regression testing of the API. This enables mitigation of the risks noted about testing with mock services but where deployment of the real ones can take too long or be too resource intensive to perform on a per-commit basis. In addition, tests such as DAST or fuzzing could be run indefinitely. Moving these to a nightly or weekly pipeline could enable a longer minimum time for these tests to run while allowing a fast developer workflow.

6.3 Testing Skill Sets

As mentioned in Test Environments (Section 6.2), when constructing a system verification or test plan, it is important to ensure that the appropriate laboratory or range environments are available. Equally important is having the correct personnel with the right skills to execute the planned testing. The need to track and manage cyber-skilled engineers is described in DoD Directive 8140.01 (2020). That directive and DoD Instruction 8140.02 (2021) provide the basis for the DoD Cyber Workforce Framework. This framework offers information about the roles needed for performing cyberspace functions, including knowledge, skills, and abilities. The specific skills needed for performing API testing depend on the specific technology stack and deployed environment the API uses.

While many skill sets are needed for testing an API or any IT system, it is important to realize that this testing should be continuous when following a DevSecOps model. Therefore, all team members must be skilled in testing since it is a critical activity. The ISO 2675-2021 standard for DevOps (2021) describes the verification process and continuous testing as follows:

> *In DevOps, the Verification process … involves developers, testers, user representatives, and other related stakeholders, including QA [Quality Assurance]. Continuous verification through the DevOps pipeline shall confirm that the system performs as required. Continuous testing is the practice of interweaving and automating various testing activities into software planning, development, delivery, and deployment processes.*

Given that the pace of development in a DevSecOps project should approach continuous delivery (CMU SEI 2022), it is necessary to perform testing as often as possible and in sync with the development cycle. Enabling continuous testing using a Lean, Agile, or Scrum framework can conflict with the complete set of skills required. The current DoD Cybersecurity Test and Evaluation Guidebook (DoD February 2020) proposes 26 possible test and evaluation roles as part of a system cybersecurity working group. For many projects, it might be infeasible to staff this many roles or include them all as members of the development team. Each role, however, is applicable to API test and evaluation; therefore, a project or program developing an API must plan to engage dedicated or specialized test personnel in addition to conducting routine automated testing. Each testing role brings a different perspective to the design and building of the automated tests and also should be engaged when tests fail.

Test personnel, developers, and all other members of an Agile team should receive training in the Agile methodology, practices, and specific ways the team or program has tailored their own implementation. This training should help to promote cross-functional teams with all members pulling work (including testing) from a backlog. When it comes to cross-functional teams, for testing, software developers are not acceptable substitutes for actual API users. Therefore, it is important to have user representation on the development team or to test using actual end users. Complications can include intricacies in actual workflows as opposed to policies or procedures handed down as requirements. API users' domain knowledge is therefore a required skill set when testing APIs.

6.4 Contract Testing

APIs are useful tools for decoupling software and allowing both modular construction and testing. Using APIs can allow multiple teams to work on different components of a software system independently. If each component talks to the other through a well-defined API, it can ease the burden of integration testing when the system is large enough to make it difficult to

assemble all the needed operators and testers to perform a full end-to-end test. An example of such a system is a military satellite communications system. It is not reasonable to test the system only once the satellites are in orbit and users are deployed to the area of operations intended to be covered by the system.

In a general sense, an API is a contract between a software application's provider and consumer that specifies what the system will do. In 1986, Bertrand Meyer coined the idea of programming by contract or design by contract (Meyer 1997, 342). In this paradigm, a software engineer formally defines a specification of each function or method that the system exposes. These specifications include noting preconditions, postconditions, and invariants (Kutztown 2014). Preconditions are the criteria required before a function is executed. These are things that the service or API provider expects to be true before a function is called. Postconditions are the state or set of criteria that must be true after a function is executed. Invariants are the data or states that, regardless of the operation or transformation applied by the function, will not change.

The benefits of the formal specification of a contract when testing an API are twofold:

- Before any implementation of an API is provided, a well-defined API contract makes it possible for the development team to write example client code that uses the functions or endpoints. This API user testing eliminates any time spent implementing irrelevant methods or API calls that are hard to use from the client perspective.
- During integration testing of different software modules, some systems can be hard to simulate in the test environment, the target system can be expensive to operate on (e.g., quantum computing hardware) (IBM Qiskit 2024), or the target system is not even built yet. In these cases, having an API contract that accurately represents a software module or library can help the software's producers and consumers conduct the integration testing.

How contracts are defined in a particular language varies. In Java, using Javadoc comments to document the parameters, return values, exceptions, and function's purpose is a common (though less formal) way of documenting a contract. A variety of tools and programming languages offer varying levels of formality for defining contracts that can help enable the verification of API usage.

The use of the OpenAPI (Swagger 2021) specification to define an API aligns well with the contract programming paradigm of specifying preconditions, such as the domain of inputs and a description of the endpoint. OpenAPI also allows a contract developer to specify the return from an endpoint, including the return code, a description of what that return code means, the schema of any returned data, and examples of the data. OpenAPI potentially falls short in its ability to specify invariants of a function. For instance, in a REST API, GET requests should be

idempotent; however, there is no way to document what an endpoint may or may not change in terms of states outside of a text description.

6.5 AI-Enabled Testing

While a nascent field, AI-enabled testing of all software, APIs included, is growing. The following sections discuss some more traditional methods as well as some of the latest research in AI (e.g., natural language processing (NLP)), ML (e.g., supervised learning), as they are being used for API testing. This field of research is nascent, and as more tools come online it will be another option for testing. The general guidance is to start with more traditional, proven methods of testing as described in this document and elsewhere and then to move into these newer methods as needed for a given program or project.

Several types of relatively traditional testing such as random argument testing or fuzzy logic for assertion-based testing, can be enhanced through the use of AI or ML (Alakeel 2015).

6.5.1 Test Generation

One proposed technique is using NLP, such as OpenAPI, to enhance API specifications (Kim 2023). These plain-text descriptions often provide examples, parameters, and input domain information, and some might infer new rules that have not been documented. Researchers extracted these descriptions via an NLP model and parsed them for additional API rules that were not previously written. These new rules are validated against the document and injected as new rules. This novel method could be wrapped up as a validation stage in a CI pipeline to ensure every API change provides standardized, accurate specifications.

In another case, researchers used a large language model (LLM) to generate test data for an API successfully from an OpenAPI specification. The researchers thoughtfully used prompt engineering to generate useful response data sets. The researchers explained, “We design[ed] a prompt specific to the testing endpoint and status code being evaluated. This prompt includes relevant information about the scenario, such as the API endpoint, the expected status code, and any additional instructions for generating suitable test data or verifying the response” (Nguyen, et al. 2023). Applying LLM in this way can be helpful for finding corner cases where existing tools, such as fuzzers, may fail to find them.

6.5.2 Verifying Correctness

With AI-enabled testing, there is a need to verify the correctness of the output of the models. Research has shown that foundational[5] models can fail to identify well-understood errors in code

[5] Large language models trained on a diverse data set and not fine-tuned to a specific domain.

(Sherman 2024). In addition, as with any test method, there is a need to integrate the use of the tool or method into the existing test pipeline. For example, Cornell University researchers have proposed a process for LLMs called Assured LLM-Based Software Engineering (Alshahwan, Automated Unit Test Improvement using Large Language Models 2024b). This strategy puts some guardrails on the generation of code by models that safeguard against regression and ensure a measurable enhancement in code coverage. This is especially important for API testing because regressions in API capabilities are the most visible to the product's end user. With this set of guarantees in place, commercial companies have improved the deployment of their systems using these types of tests (Alshahwan, Assured LLM-Based Software Engineering 2024a); however, they are still in the early experimental phases of using this strategy and are targeting improvement of their test cases, not replacing the foundational work of writing tests.

6.6 Zero Trust Testing

To consider the nexus of ZT, ZTA and API testing, this guide uses five of the seven basic tenets of ZT from NIST SP 800-207 (2020). These tenets lead to several conclusions regarding ZT, ZTA and APIs:

1. API Authentication and Authorization

Continuing to move up the stack from configuration and infrastructure, a team testing an API should next look for common broken authentication issues within the API. Broken authentication is the second of the top 10 security risks to APIs according to OWASP (Yallon 2023). This type of testing should occur in unit, integration, and application tests, depending on how authentication is integrated with the API. Tests should focus on finding flaws in an API's implementation that would allow a user to assume the identity of someone else; this is where testing for *non-repudiation* is useful.

Broken authorization is another dimension that needs to be tested in ZT architectures. Authorization verifies that the user has the right permissions to use a resource. With ZT, users should be given the minimum permissions to accomplish the intended function and no more (i.e., the principle of least privilege). When testing an API for broken authorization, testers should try to find flaws in what a user is able to request or modify through the API. For example, a tester can determine whether an attacker can modify the ID of an object that is sent in an API request, which indicates a broken object level authorization (BOLA) (Yallon 2023). These types of tests can generally be handled at the unit level, but they might require integration testing if the API offloads the authentication and authorization to a central service, which is a recommended best practice (Yallon 2023).

2. API Replay Attack Vulnerabilities

Replay attacks, in which a malicious actor reuses an expired credential, are similar to broken authentication. Testing against this type of vulnerability ensures that any tokens or keys used are short-lived and valid only for connecting to the API in the current session.

3. Securing APIs with ZT

Testing that the data exchanges between an API and its users are secured (e.g., through encryption) is specialized testing that requires a particular skill set. As with authentication and authorization, the accepted best practice is to use libraries and functions provided for this purpose and not implement them from scratch. To verify that an API is appropriately implementing a secure connection falls into the integration testing category and may require capturing network traffic between the user and the API to ensure that a secure handshake was made and the correct protocol options were used. This type of security testing, while necessary for ensuring compliance with ZT, is admittedly specialized and may require dedicated personnel trained in this area.

4. API Testing and Development Environments are Assets.

Developers must have an infrastructure in place to run tests and pipelines to enable the testing process. Testing, preproduction, and other environments must be accounted for and interoperate with the tools that enforce ZT (Sanders 2021).

5. API Monitoring

APIs provide a unique opportunity to enable the monitoring of the systems where they are deployed. When considering a ZT architecture, it is important that, once the system is in place, it be continuously evaluated for security-policy violations. In ZT there is not a single set of rules defined once. ZT security policy is context aware and adaptable (NIST SP 800-207 2020, 3.3.1). As such, it may be necessary to pull different types of information from different parts of an information system to determine if security has been compromised. APIs can enable this determination by allowing access to the information about API access, including who accessed what and when. This ability to log user activity should be considered in the design phase, but it also should be exercised in the testing phase.

7 Conclusion

This document has underscored the significance of existing APIs in supporting improved interoperability in DoD systems. APIs are essential to interoperability, facilitating data sharing and integration between diverse systems and applications. As technology evolves, so will APIs, bringing enhanced capability and functionality. Embracing these future technologies will not only enhance the DoD's agility, efficiency, and effectiveness but also will empower U.S. warfighters with the most advanced tools and information to succeed in their missions.

Adopting APIs will be challenging. A concerted effort of clear communications, comprehensive training, and active stakeholder engagement will help the Department to overcome cultural and other challenges for enhanced joint mission interoperability. Success may require changes to organizational constructs, acquisition processes, and the acquisition pathways. It is vital to involve all stakeholders, from leadership to developers and end-users. A culture of openness and collaboration will help ensure programs develop future systems equipped to leverage APIs for enhanced interoperability.

While enabling access to sensitive data and functionalities, APIs can be potential targets for malicious actors. The DoD is prioritizing robust security measures, such as authentication, authorization, encryption, and continuous monitoring, to safeguard the confidentiality, integrity, and availability of its APIs. Privacy considerations data anonymization and consent management with regard to warfighter and non-warfighter personal information should be integral to API design and implementation to protect this vital information.

This document underscores the need for the DoD to proactively embrace future technologies while effectively managing the cultural shift required for their adoption. By recognizing the potential of future technologies, implementing robust change management strategies, and prioritizing security and privacy, the DoD can successfully adopt APIs. This effort will not only enhance operational capabilities but also will help ensure warfighters and programs developing future systems are equipped with the most advanced and interoperable tools to achieve their mission objectives.

Appendix A: API Project Governance Considerations

The API project team should consider the following factors (Table A-1) when designing or updating an API framework, ecosystem, model or standard for DoD use.

Table A-1. API Project Governance Considerations

Factor	Consideration
API Strategy	The API project should communicate its approach to interoperability prior to starting system development. The intent of this document is to establish a vision and initial plans to facilitate understanding by stakeholder. This includes envisioned near-term capability and possible evolutions for the future. This would also include a definition of scope of the API, describing what would be considered valid solution space for the API now and in the future. This strategy should be updated as the program evolves.
Use Cases	The API project should provide comprehensive use case descriptions and diagrams of how the API is intended to be used. This allows the potential consumers to validate if the API meets the intended requirements. This may not preclude them from using it in new ways but is another way to describe how the API structure came to be what it is and more quickly understand its design. This also describes the problem space chosen for the API in the strategy and allows current and future stakeholders a way to communicate new and unforeseen needs.
API Contracts	The API project should describe Service Life Agreement (SLA) rules that should be followed for use of the API including Non-Functional Requirements (NFRs). The API contract should also provide API usage description information including inputs and outputs (Sindall 2020).
Hosting	The API project should describe how the API services are to be hosted and their required reliability.
API Project Performance and Design	The API project should describe what performance criteria the API functions and responses must meet.
Registry and Discovery	The API project should describe how the API discovery process will work. This will include a discussion of service catalog, registry categorization, and interaction styles (e.g., REST, stateful).
Scalability	The API project should describe scalability requirements.
Transport	The API project should describe what type of transport services the API will use. These descriptions include the required network communication frameworks (e.g., HTTPs, FTPs); data serialization (e.g., XML, JSON, ASN.1, etc.); and network confidentiality, integrity, and nonrepudiation approaches.

Factor	Consideration
Security	The API project should describe the general security framework in which the API resides and what parts of the security framework need to be instituted. This at least includes the required authentication, authorization, and “need to know” checks.
Design	The API project should describe how stakeholder’s inputs are taken into account for the project during the initial and follow-up updates. Also describe the API design patterns, caching requirements, data retrieval function, and data semantics. Information models used and fault tolerant flows (Sindall 2020).
Quality Reviews	The API project should describe how API ecosystem design and implementation quality reviews will be done to ensure the API meets the API standard and has the desired attributes. Ensure governance rules are met before deployment (Sindall 2023).
Testing	The API project plan should describe how the API is tested prior to being deployed as well as the processes used to quickly and consistently test updates via automation to the API. This may include automated governance checks (Sindall 2020).
Deployment	The API project should describe the deployment approach such as API Library versus several libraries/formats. Also, describe the registration, setup, configuration sequences that allow quick onboarding.
Feedback	The API project should describe how stakeholders and developers can provide improvements, suggestions, report issues, and raise concerns about API not performing as documented.
Versioning and Backward Compatibility	Establishing a versioning and backward compatibility plan is critical for ensuring changes to the APIs do not break existing integrations. Implementing a comprehensive versioning strategy from the outset provides a clear upgrade path for API consumers and avoids any workflow disruptions. One key consideration is the use of semantic versioning, a widely adopted three-part version numbering scheme (i.e., major.minor.patch). Major version changes indicate significant changes that may break backward compatibility. Minor version changes indicate new features or functionality that are backward compatible. Patch version changes indicate bug fixes or minor updates that are fully backward compatible. Use of this scheme communicates API changes and provides predictable upgrade paths for users. Thus, existing integrations will continue to function as expected, while also allowing for the introduction of new features and functionality. Another key consideration is planning for backward compatibility. Backward compatibility ensures that existing integrations continue to function correctly, even when changes are made to the API. This can be achieved by maintaining existing endpoints, providing fallback mechanisms, or implementing versioning strategies that allow for multiple versions of the API to coexist. Planning for both versioning and backward compatibility from design to implementation ensures that the API remains stable, reliable, and functional over time. This builds trust and confidence with API users and ensures that the API continues to meet their needs and expectations.

Factor	Consideration
Updating	The API project should describe how an API framework is updated and issue any new release of the updated API implementation. This creates a robust API versioning approach (Sindall 2023).
Deprecating	The API project should describe how the API should handle planning and timing of removing aspects of the API that are no longer desired or have been replaced by improved functionality.
Tracking Use	The API project should describe how the project can track use of the API, if applicable. Where applicable this can help create new use case and business case support for the changes.
Telemetry	For further information on metrics to track performance, scalability, security, and tracking use, see metrics section in Design and Implementation section.

Appendix B: Common API Vulnerabilities and Threats

The following are some of the most common API vulnerabilities and threats:

- Injection Attacks - Injection attacks occur when an attacker sends malicious input to an API with the intent of executing unauthorized commands or accessing sensitive data. Common types of injection attacks include SQL injection, XML injection, and command injection. Injection attacks can be particularly dangerous in the context of the DoD, as they can be used to gain unauthorized access to sensitive systems and data.
- Cross-Site Scripting (XSS) Attacks - XSS attacks occur when an attacker injects malicious code into a web page or API response, which is then executed by a user's browser. This can allow the attacker to steal sensitive data or perform unauthorized actions on behalf of the user. XSS attacks can be particularly dangerous in the context of the DoD, as they can be used to compromise user accounts and gain access to sensitive systems and data.
- Denial-of-Service (DoS) and Distributed Denial of Service (DDoS) Attacks - DoS attacks occur when an attacker floods an API with requests in an attempt to overwhelm the system and prevent legitimate users from accessing it. DoS and especially DDoS attacks can be particularly damaging in the context of the DoD, as they can disrupt critical systems and services.
- Insufficient Authentication and Authorization - Insufficient authentication and authorization can occur when an API does not properly verify the identity of users or restrict access to sensitive data and systems. This can allow unauthorized users to access sensitive data and systems, potentially leading to data breaches and other security incidents.
- Insecure Data Storage - Insecure data storage can occur when an API stores sensitive data in an unencrypted or otherwise insecure manner. This can allow attackers to steal sensitive data, such as passwords and other credentials, and use it to gain unauthorized access to systems and data.
- XML External Entity (XXE) Attacks - XXE (XML External Entity) attacks are a type of injection attack that can be used to exploit vulnerabilities in APIs that process XML data. XXE attacks are a significant API vulnerability and threat because they can be used to gain unauthorized access to sensitive data and systems. In the context of the DoD, XXE attacks can be particularly dangerous as they can be used to compromise critical systems and services. To mitigate the risk of XXE attacks, it is important to implement strong API security measures, such as input validation to prevent injection attacks, and to use secure XML parsers that are not vulnerable to XXE attacks.

- Insecure Data Transmission - Unsecure data transmission is an API vulnerability and threat that occurs when data is transmitted over a network in an unencrypted or otherwise insecure manner. This can allow attackers to intercept and read sensitive data, such as passwords and other credentials, and use it to gain unauthorized access to systems and data. To mitigate the risk of unsecure data transmission, it is important to use strong encryption mechanisms, such as TLS (Transport Layer Security), to protect data in transit.
- Improper Access Controls and Authorization Flaws - Improper access controls and authorization flaws are an API vulnerability and threat that occur when an API does not properly restrict access to sensitive data or functionality. This can allow attackers to gain unauthorized access to systems and data, potentially leading to data breaches and other security incidents. To mitigate the risk of improper access controls and authorization flaws, it is important to implement strong access controls and authorization mechanisms, such as role-based access control (RBAC) and attribute-based access control (ABAC).
- Security Misconfigurations and Improper Error Handling - Security misconfigurations and improper error handling are an API vulnerability and threat that occur when an API is not properly configured or when errors are not handled in a secure manner. This can allow attackers to exploit vulnerabilities in the API and gain unauthorized access to systems and data. To mitigate the risk of security misconfigurations and improper error handling, it is important to implement strong security configurations and to properly handle errors in a secure manner.
- Insider Threats and Unauthorized Access - Insider threats and unauthorized access are API vulnerabilities and threats that occur when individuals with authorized access to an API misuse their privileges or when unauthorized individuals gain access to the API. This can lead to unauthorized disclosure, modification, or destruction of sensitive data, as well as disruption of services. To mitigate the risk of insider threats and unauthorized access, it is important to implement strong access controls, such as RBAC and least privilege principles. Regular monitoring and auditing of API activities can also help detect and prevent unauthorized access. In addition, implementing strong authentication mechanisms, such as multi-factor authentication (MFA), can further enhance security and protect against unauthorized access.

Appendix C: API Security Challenges

C.1 Injection Attacks and Their Impact on Mission-Critical Systems

Injection attacks pose a significant threat to mission-critical systems within the DoD context. These attacks involve the introduction of malicious data or code into a system, exploiting vulnerabilities to manipulate system behavior, compromise data integrity, or gain unauthorized access. In the DoD context, where mission-critical systems are integral to intelligence, command and control of military forces, weapons systems, and fulfilling military requirements, the impact of injection attacks can be severe. They can disrupt operations, endanger operator safety, compromise sensitive information, and potentially jeopardize national security. The DoD's cybersecurity initiatives aim to mitigate such threats through secure coding practices, automated security testing, and continuous monitoring; however, the evolving nature of injection attacks and the complexity of DoD systems present ongoing challenges.

C.2 Authentication and Authorization Issues in a Multi-Domain Environment

Authentication and authorization in a multi-domain environment within the DoD context present unique security challenges. Authentication, the process of verifying the identity of a user, device, or system, and authorization, the process of granting or denying access rights to resources, are critical for maintaining the security and integrity of DoD systems. In a multi-domain environment, where resources and users are distributed across various domains, ensuring consistent and secure authentication and authorization becomes complex. This complexity can lead to potential vulnerabilities, such as unauthorized access or privilege escalation. The DoD addresses these challenges through robust MFA, RBAC, and ABAC mechanisms, along with continuous monitoring and auditing; however, the dynamic nature of multi-domain environments and the evolving threat landscape continue to pose significant challenges.

C.3 Data Breaches and Protection of Sensitive Information

Data breaches and the protection of sensitive information are significant security challenges within the DoD context. The DoD manages vast amounts of sensitive data, including classified military information, personnel records, and intelligence data. Data breaches can lead to the exposure of this sensitive information, with potential impacts on national security, operational effectiveness, and the privacy of personnel. The DoD has experienced significant data breaches in the past, highlighting the importance of robust data protection measures. These measures include data encryption, secure data handling practices, and continuous monitoring for potential threats. However, the complexity of the DoD's information systems, the sophistication of adversaries, and the evolving nature of threats continue to pose challenges to the protection of sensitive information within the DoD.

C.4 Service Discovery Threats

The service discovery threats outlined in the "Service Discovery Threat Model for AD HOC Networks" by Adrian Leung and Chris Mitchell highlight security challenges within the DoD context. Ad hoc networks, which are dynamic and vulnerable, present unique security and privacy challenges. Service discovery, the process of finding and connecting to available services, is particularly susceptible to threats in these networks. The DoD relies on secure and reliable service discovery mechanisms to ensure the availability and integrity of critical services. However, the dynamic nature of ad hoc networks and the potential for malicious actors to exploit vulnerabilities in service discovery protocols pose significant challenges to the DoD's ability to maintain secure and resilient communication and information exchange. Implementing robust security measures, such as encryption, authentication, and intrusion detection systems, is crucial to mitigating these threats and ensuring the security of DoD operations in ad hoc network environments (Leung and Mitchell 2023).

Other related threats include:

- Service Spoofing
- Passive Listening
- Data Alteration

C.5 Cascading Failure

Cascading failure, a phenomenon where the failure of one component triggers a chain reaction of failures in interconnected systems, poses significant cybersecurity challenges within the DoD. In the context of API cybersecurity threats, cascading failures can occur when a vulnerable API is exploited, leading to the compromise of other interconnected APIs or systems. This can have severe consequences for the DoD, as it relies on a complex network of interconnected systems and APIs to support critical operations. The potential for cascading failures highlights the importance of implementing robust security measures, such as secure coding practices, vulnerability scanning, and continuous monitoring, to prevent and mitigate the impact of API cybersecurity threats and ensure the resilience and integrity of DoD systems.

Glossary

Unless otherwise noted, the following definitions are suggested by the authors of this guide for their relevance to APIs. The definitions are not intended to be authoritative in all contexts.

Term	Description	Source
abstraction	The process of simplifying complex systems or concepts by focusing on essential features while hiding unnecessary details.	https://www.imedpub.com/articles/abstraction-simplifying-complexity-in-software-engineering.php?aid=50800#:~:text=Abstraction%20is%20a%20fundamental%20concept%20in%20software%20engineering%20that%20helps,and%20facilitating%20scalability%20and%20maintainability
academia	The world of education and research, typically associated with universities and scholarly activities.	
access controls	Security measures and policies that determine who is allowed to access or modify certain resources or data.	
acquisition	The process of obtaining or procuring something, often used in the context of acquiring assets or technology.	
Adaptive Acquisition Framework	A set of acquisition pathways to enable the workforce to tailor strategies to deliver better solutions faster.	https://aaf.dau.edu/
acquisition pathways	The various routes or methods used to obtain resources or technology, typically within a business or organizational context.	
acquisition processes	The procedures and steps involved in acquiring resources, technology, or assets, often including planning, procurement, and implementation.	
algorithm	A procedure for solving a mathematical problem (as of finding the greatest common divisor) in a finite number of steps that frequently involves repetition of an operation.	https://www.merriam-webster.com/dictionary/algorithm

Term	Description	Source
application programming interface (API)	A set of definitions and protocols for building and integrating application software.	https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces
API gateway	A data-plane entry point for API calls that represent client requests to target applications and services. It typically performs request processing based on defined policies, including authentication, authorization, access control, SSL/TLS (Secure Sockets Layer/Transport Layer Security) offloading, routing, and load balancing.	https://www.nginx.com/learn/api-gateway/#:~:text=An%20API%20gateway%20is%20a%20data%2Dplane%20entry%20point%20for,%2C%20routing%2C%20and%20load%20balancing
API-first	Prioritizing the APIs that support your application and focusing on the value they can deliver to your business, rather than just scrambling to deliver a single application and creating an API as an afterthought.	https://www.postman.com/api-first/#:~:text=Being%20API%2Dfirst%20means%20prioritizing,an%20API%20as%20an%20afterthought
application aware health checking	An API monitoring method that checks your API and alerts you when it notices something is amiss. A diagnostic tool for your code base that can help you find problems before they become more significant headaches than they need to be.	https://testfully.io/blog/api-health-check-monitoring/
architects	Professionals responsible for designing the overall structure and organization of software systems or IT infrastructure.	
artificial intelligence (AI)	A machine's ability to perform the cognitive functions usually associated with human minds.	https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-ai
attack detection	Detecting suspicious API traffic. Attack detection is of utmost importance in today's digital landscape. With the increasing reliance on APIs for data exchange between different applications and systems, it has become crucial to ensure the security and integrity of these interactions.	https://nonamesecurity.com/blog/how-to-detect-suspicious-api-traffic/
authentication	Carefully and comprehensively identifying all related users and devices. Typically requires client-side applications to include a token in the API call so the service can validate the client.	https://www.techtarget.com/searchapparchitecture/tip/10-API-security-guidelines-and-best-practices

Term	Description	Source
backward compatibility	The ability of a RESTful API to handle requests from the clients that use an older version of the API without breaking or returning errors. Reduces the friction and cost for the clients to upgrade to the new version of the API. Also preserves the trust and reliability of the web service, as the clients can expect the API to work as intended.	https://www.linkedin.com/advice/0/how-do-you-design-restful-api-supports#:~:text=Backward%20compatibility%20is%20the%20ability,new%20version%20of%20the%20API
canary release	A technique to reduce the risk of introducing a new software version in production by slowly rolling out the change to a small subset of users before rolling it out to the entire infrastructure and making it available to everybody.	(Sato 2014) https://martinfowler.com/bliki/CanaryRelease.html
circuit breakers	Wrapping a protected function call in a circuit breaker object, which monitors for failures. Once the failures reach a certain threshold, the circuit breaker trips, and all further calls to the circuit breaker return with an error, without the protected call being made at all. Usually involves a monitor alert if the circuit breaker trips.	(Sato 2014) https://martinfowler.com/bliki/CircuitBreaker.html
code review	The act of consciously and systematically convening with one's fellow programmers to check each other's code for mistakes; shown to accelerate and streamline the process of software development as few other practices can.	https://smartbear.com/learn/code-review/what-is-code-review/
common data model (CDM)	Contains a uniform set of metadata, allowing data and its meaning to be shared across applications. In addition to the uniform metadata, a CDM includes a set of standardized, extensible data schemas that include items such as entities, attributes, semantic metadata, and relationships. Once all the elements of the CDM are defined, methods to access and operate on the data are developed so all applications can use these same, standardized procedures.	https://www.synopsys.com/glossary/what-is-common-data-model.html
communication frameworks	Software libraries or protocols that facilitate communication and data exchange between software components or systems.	
confidentiality	The principle of protecting sensitive or confidential information from unauthorized access or disclosure.	
Conway's Law	Essentially the observation that the architectures of software systems look remarkably similar to the organization of the development team that built it.	https://martinfowler.com/bliki/ConwaysLaw.html

Term	Description	Source
cross-site scripting (XSS)	A type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user.	https://owasp.org/www-community/attacks/xss/
cross-site request forgery (CSRF)	An attack that forces an end user to execute unwanted actions on a web application in which they are currently authenticated.	https://owasp.org/www-community/attacks/csrf
(Agile) culture	At an enterprise level, moving strategy, structure, processes, people, and technology toward a new operating model by rebuilding an organization around hundreds of self-steering, high-performing teams supported by a stable backbone.	https://www.mckinsey.com/capabilities/people-and-organizational-performance/our-insights/doing-vs-being-practical-lessons-on-building-an-agile-culture
cybersecurity (API)	Strategies and solutions to understand and mitigate the vulnerabilities and security risks of Application Programming Interfaces (APIs). APIs are a critical part of modern mobile, software as a service (SaaS), and web applications and can be found in customer-facing, partner-facing and internal applications. By nature, APIs expose application logic and sensitive data such as Personally Identifiable Information (PII) and because of this have become a target for attackers. Without secure APIs, rapid innovation would be impossible.	https://owasp.org/www-project-api-security/
data anonymization	The process of removing particular pieces of private information that could be used to identify a person in data.	https://www.splunk.com/en_us/blog/learn/data-anonymization.html
data fabric	An architecture and set of data services that provide consistent capabilities across a choice of endpoints spanning hybrid multicloud environments. A powerful architecture that standardizes data management practices and practicalities across cloud, on premises, and edge devices.	https://www.netapp.com/data-fabric/what-is-data-fabric/
data mesh	Principles to help address changes in the data landscape and speed of response to change: (1) domain-oriented decentralized data ownership and architecture, (2) data as a product, (3) self-serve data infrastructure as a platform, and (4) federated computational governance.	https://martinfowler.com/articles/data-mesh-principles.html
data semantics	The meaning and interpretation of data, often defined through metadata and ontologies.	

Term	Description	Source
data serialization	The process of converting structured data into a format suitable for transmission or storage, such as JSON or XML.	
data visualization	The presentation of data in graphical or visual formats to facilitate understanding and analysis.	
data-centric	Focusing on the data itself as the central element in system design and decision-making.	
deprecating	Phasing out or marking a software feature or API as obsolete, typically with the intention of removing it in future versions.	
developmental test and validation	The process of verifying whether the specific requirements to test development stages are fulfilled, based on solid evidence. In particular, test validation is an ongoing process of developing an argument that a specific test, its score interpretation, or use is valid.	https://assess.com/test-validation/#:~:text=Test%20validation%20is%20the%20process,interpretation%20or%20use%20is%20valid
DevSecOps	A software engineering culture that guides a team to break down silos and unify software development, deployment, security, and operations. Success in adopting DevSecOps requires buy-in from all stakeholders, including: leadership, acquisition, contracting, middle-management, engineering, security, operations, development, and testing teams. Stakeholders across the organization must change their way of thinking from “I” to “we,” while breaking team silos, and understanding that the failure to successfully deliver, maintain, and continuously engineer software and its underlying infrastructure is the failure of the entire organization, not one specific team or individual.	https://dodcio.defense.gov/Portals/0/Documents/Library/DevSecOps%20Playbook_DoD-CIO_20211019.pdf
digital modernization	The process of updating and adapting an organization’s digital systems and technologies to meet current and future needs.	
ecosystem	A community of interconnected components or entities that interact and influence each other, often used in the context of software or technology.	
emerging technology	New and cutting-edge technologies that have the potential to disrupt existing industries or create new opportunities.	

Term	Description	Source
encoding attack	An attack in which malicious data or scripts are embedded in input to exploit vulnerabilities in a system.	
encryption	The process of converting data into a secure and unreadable format to protect it from unauthorized access.	
enterprise standards	Established guidelines, protocols, and best practices that an organization follows to ensure consistency and interoperability across its systems.	
error handling	The process of identifying, reporting, and managing errors or exceptions in software to ensure robustness and graceful degradation.	
experimentation	The systematic testing and exploration of new ideas, features, or solutions to gather data and make informed decisions.	
fault tolerant	The ability of a system to continue functioning or provide degraded service in the presence of faults or failures.	
filtering	The process of selecting or extracting specific data or information from a larger data set based on predefined criteria.	
firewall	A network security device or software that monitors and controls incoming and outgoing network traffic to protect against unauthorized access or threats.	
frameworks	Predefined structures and libraries that provide a foundation for building software applications.	
functional testing	Testing that focuses on verifying that the software functions according to specified requirements and performs its intended tasks.	
fuzz testing	Also called fuzzing, an automated software testing method that injects invalid, malformed, or unexpected inputs into a system to reveal software defects and vulnerabilities. A fuzzing tool injects these inputs into the system and then monitors for exceptions such as crashes or information leakage. Fuzzing introduces unexpected inputs into a system and watches to see if the system has any negative reactions to the inputs that indicate security, performance, or quality gaps or issues.	https://www.synopsys.com/glossary/what-is-fuzz-testing.html

Term	Description	Source
(API) governance	The processes and controls implemented to manage, monitor, and maintain APIs (application programming interfaces). It involves defining standards, policies, and guidelines for API design, development, deployment, and usage. The goal of API governance is to ensure consistency, security, scalability, and reliability across all API.	https://nonamesecurity.com/learn/what-is-api-governance/
Hypertext Transfer Protocol (HTTP)	A fundamental protocol used for communication on the World Wide Web. It defines the rules and conventions for transferring text, images, videos, and other resources between web servers and web clients (typically web browsers). HTTP operates as a request-response protocol, in which a client (usually a web browser) sends an HTTP request to a web server, and the server responds with the requested content or an error message.	
HTTP verb tampering	An attack in which an attacker manipulates the HTTP request method (HTTP verb) to perform unauthorized actions on a web application.	
identity and access management	The practices and technologies used to manage and secure digital identities and control access to resources.	
implementation	The process of translating a design or plan into a working system or application.	
industry	A specific sector of economic activity, characterized by similar products, services, and business practices.	
information model	A conceptual representation of data and its relationships within a system or domain.	
injection attack	An attack in which malicious code or input is injected into an application, potentially leading to data breaches or system compromise.	
integration	The process of combining different software systems or components to work together as a unified whole.	
integrity	The assurance that data or resources have not been altered or tampered with in an unauthorized or malicious manner.	
intelligence analytics	The process of collecting, analyzing, and interpreting data to gain insights and make informed decisions in intelligence and security contexts.	

Term	Description	Source
interface	A point of interaction between different software components, allowing them to communicate or exchange data.	
interoperability	The ability of different systems or components to work together and exchange data seamlessly.	
Joint	In a military context, collaborative efforts involving multiple branches or Services.	
landscape	The overall view or context of a situation, often used to describe the broad environment in which a system operates.	
lateral movement within network	A tactic used by attackers to move horizontally within a network after gaining initial access, often to escalate privileges or reach valuable targets.	
legacy system	An older or outdated computer system, software, or technology that may still be in use but is typically less efficient or secure.	
load balancing	The distribution of network traffic or computing workloads across multiple servers or resources to ensure optimal performance and reliability.	
logging	The process of recording events, actions, or transactions in a system for monitoring, troubleshooting, and auditing purposes.	
logistics	The planning, management, and coordination of resources, often in the context of supply chain management.	
machine learning (ML)	A subset of artificial intelligence (AI) that involves the development of algorithms and models that allow computers to learn from data and make predictions or decisions.	
maintaining	The ongoing process of keeping a system or application operational and up to date.	
mental lock-in	A cognitive bias in which individuals become overly committed to a particular idea or approach, making it challenging to consider alternative solutions.	

Term	Description	Source
metrics	Quantifiable measures used to assess and evaluate the performance, effectiveness, or quality of a system or process.	
micro-segmentation	Network security strategy that divides a network into smaller, isolated segments to improve security and control.	
modeling and simulation	The use of mathematical models and computer simulations to replicate real-world processes or systems for analysis and experimentation.	
monitoring	The continuous observation and tracking of a system's performance, behavior, or security.	
networked weapons	Military or defense systems that are interconnected and can communicate with each other for coordinated operations.	
nonrepudiation	The assurance that a user cannot deny the authenticity or origin of a message or action they initiated.	
open standards and protocols	Publicly available and widely accepted specifications for communication and data exchange.	
operational	Related to the day-to-day activities and functions of an organization or system.	
organization constructs	The structural elements, roles, and relationships within an organization.	
pagination	The practice of dividing large sets of data or content into smaller, manageable pages for display or retrieval.	
penetration testing	A security assessment where ethical hackers attempt to identify vulnerabilities in a system by simulating real-world attacks.	
performance	The ability of a system to execute tasks efficiently and meet specified criteria.	
platform	A software or hardware environment that provides a foundation for building and running applications.	
policy	A set of rules, guidelines, or principles that dictate decisions and actions within an organization or system.	

Term	Description	Source
proactive threat detection	The practice of identifying and mitigating security threats before they can cause harm or damage.	
program managers	Individuals responsible for overseeing and managing projects or initiatives within an organization.	
providers	Entities that offer services, resources, or solutions to others.	
rapid prototyping	The process of quickly creating a working model or prototype of a product or system to test and validate concepts.	
real-time systems	Systems that operate and respond to events immediately as they occur.	
recursive	A process or function that calls itself to solve a problem by breaking it down into smaller, similar tasks.	
Representational State Transfer (REST)	An architectural style for designing networked applications, often used with HTTP, emphasizing stateless communication and resource-based URLs.	
requirements	The specifications and criteria that define what a system or product must accomplish or include.	
Reverse Conway’s Maneuver	Adapting an organization's structure to align with desired software architecture.	
reverse proxy	A server that acts as an intermediary between client requests and one or more backend servers, often used for load balancing, security, and caching.	
scalability	The ability of a system or application to handle increased workloads or users without a significant decrease in performance.	
secure key management	The practices and processes for generating, storing, and protecting encryption keys.	
security breaches	Unauthorized access or incidents that compromise the confidentiality, integrity, or availability of data or systems.	
security compliance	Adherence to security standards, regulations, and policies to protect against security threats and vulnerabilities.	

Term	Description	Source
security posture	The overall security status and readiness of an organization or system.	
sensitive information	Data that, if disclosed or compromised, could result in harm to individuals or organizations.	
sensor fusion	Combining data from multiple sensors or sources to improve accuracy and reliability in various applications, such as navigation or surveillance.	
serialization attack	An attack that manipulates the serialization process of data to exploit vulnerabilities.	
service discovery	The process of automatically finding and identifying available network services or resources.	
Services	For the U.S. Department of Defense, the following Military Services: Army, Navy, Air Force, Marine Corps, Coast Guard, Space Force, and other supporting Components.	
session persistence	The ability to maintain session state or data across multiple interactions or requests from a user.	
social network systems	Online platforms and communities where users can connect, share, and interact with others.	
software delivery	The process of planning, developing, testing, and deploying software applications or updates.	
sorting	Arranging data or elements in a specific order, often numerical or alphabetical.	
stakeholders	Individuals or groups with an interest or concern in a project, system, or organization.	
statistical analysis	The process of analyzing data using statistical methods and techniques to draw conclusions or make predictions.	
strategic objectives	Long-term goals and plans that guide an organization’s overall direction and decision making.	
support staff	Personnel responsible for assisting users, maintaining systems, and providing technical support.	

Term	Description	Source
system of systems	A collection of interconnected and interdependent systems that work together to achieve a common goal.	
tactical	Pertaining to short-term, practical, and on-the-ground decisions and actions.	
telemetry	The remote monitoring and measurement of data, often from distant or inaccessible locations.	
thread detection and response	The identification and mitigation of threats or malicious activities in computer systems, networks, or applications.	
time to live (TTL) tokens	Tokens or data elements with a specified lifespan or expiration time.	
tracking use	Monitoring and recording how resources or services are used or accessed.	
transport systems	Infrastructure and technologies used to move people, goods, or data from one place to another.	
unauthorized access attempts	Efforts to gain unauthorized entry to a system, application, or resource.	
vendor lock-in	A situation in which a user becomes dependent on a specific vendor's products or services, making it difficult to switch to alternatives.	
versioning	The practice of assigning unique version numbers to software or data to manage changes and updates.	
vulnerabilities and threats	Weaknesses or flaws in systems, applications, or processes that can be exploited by threats or attackers.	
warfighting	The conduct of military operations and strategies in armed conflict.	
wargames	Simulated military exercises or games used for training and strategic planning.	
zero trust (ZT)	A security model that assumes no trust by default and requires strict authentication and authorization for all users and devices, regardless of their location or network access.	

Acronyms

A&A	Authentication and Authorization
ABAC	Attribute-Based Access Control
AI/ML	Artificial Intelligence/Machine Learning
API	Application Programming Interface
A&S	Acquisition and Sustainment
CDM	Common Data Model
CI/CD	Continuous Integration/Continuous Deployment
CJADC2	Combined JADC2
CSRF	Cross-Site Request Forgery
DevSecOps	Development, Security, Operations
DISR	DoD Information Technology Standards Registry
DoD	Department of Defense
DoD CIO	Department of Defense Chief Information Officer
DoS	Denial-of-Service
GW	Gateway
HTTP	Hypertext Transfer Protocol
IoMT	Internet of Military Things
JADC2	Joint All-Domain Command and Control
JSON	JavaScript Object Notation
MFA	Multi-Factor Authentication
MOSA	Modular Open Systems Approach
MVP	Minimum Viable Product
NIST	National Institute of Standards and Technology
NFR	Non-Functional Requirement

OSI	Open System Interconnection
OUSD(A&S)	Office of the Under Secretary of Defense for Acquisition and Sustainment
OUSD(R&E)	Office of the Under Secretary of Defense for Research and Engineering
PHI	Protected Health Information
PII	Personally Identifiable Information
PM	Program Manager
RBAC	Role-Based Access Control
R&E	Research and Engineering
REST	Representational State Transfer
SAST	Static Application Security Testing
SE&A	Systems Engineering and Architecture
SLA	Service-Level Agreement
SLA	Service Life Agreement
SoS	System of Systems
SSL/TLS	Secure Sockets Layer/Transport Layer Security
TTL	Time to Live (Tokens)
URL	Uniform Resource Locator
XML	Extensible Markup Language
XSS	Cross-Site Scripting

References

Alakeel, Ali M. 2015. "Using Fuzzy Logic Techniques for Assertion-Based Software Testing Metrics." *Scientific World Journal* (April). https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4427827/.

Alshahwan, Nadia, et al. 2024a. *Assured LLM-Based Software Engineering.* Cornell University. https://arxiv.org/abs/2402.04380.

Alshahwan, Nadia, et al. 2024b. *Automated Unit Test Improvement using Large Language Models.* Menlo Park, California: Meta Platforms Inc. https://arxiv.org/pdf/2402.09171.pdf.

Anderson, Ross. 2020. *Security Engineering: A Guide to Building Dependable Distributed Systems, 3rd Edition.* Wiley. https://www.wiley.com/en-us/Security+Engineering%3A+A+Guide+to+Building+Dependable+Distributed+Systems%2C+3rd+Edition-p-9781119642787.

Brady and Dianic. 2022. *Data Interoperability.* PowerPoint presentation, DoD Enterprise DevSecOps Community of Practice. https://software.af.mil/wp-content/uploads/2022/05/05-12-2022-DevSecOps-CoP-Slides-API-2.0-FINAL.pdf.

CISA. 2023. *Zero Trust Maturity Model, Version 2.0.* Cybersecurity Division, Cybersecurity and Infrastructure Security Agency (CISA), 23. https://www.cisa.gov/sites/default/files/2023-04/zero_trust_maturity_model_v2_508.pdf.

CJCSI RSI. 2019. *CJCSI Rationalization, Standardization, and Interoperability (RSI) Activities.* Chairman of the Joint Chiefs of Staff (CJCS). https://www.jcs.mil/Portals/36/Documents/Doctrine/allied_doctrine/cjcsi2700_01g.pdf?ver=fIHUSEemnzzQUMj_sjoOnQ%3D%3D.

CMU SEI. 2022. "Continuous Delivery." https://cmu-sei.github.io/DevSecOps-Model/?_gl=1*i7p6w5*_ga*MTM1NjUzMjk1My4xNzA5MTI4MDM5*_ga_87WECW6HCS*MTcwOTE0NDg1MS4xLjAuMTcwOTE0NDg1MS42MC4wLjA.#Diagrams__7b8cab04-0f58-4b71-88b6-8eef5374734e.

Computer Networking Notes. 2023. "OSI Seven Layers Model Explained with Examples." *Computer Networking Notes.* August 8. Accessed October 23, 2023. https://www.computernetworkingnotes.com/ccna-study-guide/osi-seven-layers-model-explained-with-examples.html.

Corbo, Anthony. 2023. "What Is Data Integrity?" *Built In.* January 3. Accessed October 23, 2023. https://builtin.com/data-science/data-integrity.

DepSecDef. 2021. *Creating Data Advantage.* Memorandum, Deputy Secretary of Defense (DepSecDef). https://media.defense.gov/2021/May/10/2002638551/-1/-1/0/DEPUTY-SECRETARY-OF-DEFENSE-MEMORANDUM.PDF.

DepSecDef. 2021. *DoD Software Modernization Strategy, Version 1.0.* Deputy Secretary of Defense (DepSecDef). https://media.defense.gov/2022/Feb/03/2002932833/-1/-1/1/DEPARTMENT-OF-DEFENSE-SOFTWARE-MODERNIZATION-STRATEGY.PDF.

DevSecOps Reference Architecture. 2023. *Modern Software Practices.* Chief Information Officer (CIO) Library. https://dodcio.defense.gov/library/.

DoD CIO . September 2021. *DevSecOps Playbook, Version 2.1.* Department of Defense (DoD) Chief Information Officer (CIO). https://dodcio.defense.gov/Portals/0/Documents/Library/DevSecOps%20Playbook_DoD-CIO_20211019.pdf.

DoD CIO. 2019. *DoD Enterprise DevSecOps Reference Design, Version 1.0.* DoD Chief Information Office (CIO). https://dodcio.defense.gov/Portals/0/Documents/.

DoD CIO Fundamentals Guidebook. March 2021. *DoD DevSecOps Fundamentals Guidebook: DevSecOps Tools & Activities, Version 2.0.* Department of Defense (DoD) Chief Information Officer (CIO). https://dodcio.defense.gov/Portals/0/Documents/Library/DevSecOpsTools-ActivitiesGuidebook.pdf.

DoD CIO Library. 2023. "Modern Software Practices." *Chief Information Officer (CIO) Library.* https://dodcio.defense.gov/library/.

DoD CIO. 2023. *Software Modernization Implementation Plan.* Restricted distribution, DoD Chief Information Officer (CIO).

DoD Data Strategy. 2020. *DoD Data Strategy.* Washington, D.C.: Department of Defense (DoD). https://media.defense.gov/2020/Oct/08/2002514180/-1/-1/0/DOD-DATA-STRATEGY.PDF.

DoD DevSecOps Strategy. 2021. *DoD Enterprise DevSecOps Strategy Guide.* Washington, D.C.: Department of Defense (DoD) Chief Information Officer and Under Secretary of Defense for Acquisition and Sustainment. https://dodcio.defense.gov/Portals/0/Documents/Library/DoD%20Enterprise%20DevSecOps%20Strategy%20Guide_DoD-CIO_20211019.pdf.

DoD. 2022. "DoD Announces Release of JADC2 Implementation Plan." *defense.gov.* March 17. Accessed October 23, 2023. https://www.defense.gov/News/Releases/Release/Article/2970094/dod-announces-release-of-jadc2-implementation-plan/.

DoD. February 2020. *DoD Cybersecurity Test and Evaluation Guidebook Version 2.0, Change 1.* Department of Defense (DoD). https://www.dau.edu/sites/default/files/2023-09/Cybersecurity-Test-and-Evaluation-Guidebook-Version2-change-1.pdf.

DoD. June 2023. *DoD Data, Analytics, Artificial Intelligence Adoption Strategy", Department of Defense.* Department of Defense (DoD). https://media.defense.gov/2023/Nov/02/2003333300/-1/-1/1/DOD_DATA_ANALYTICS_AI_ADOPTION_STRATEGY.PDF .

DoD Issues New Data Strategy. 2020. *defense.gov.* October. https://www.defense.gov/News/Releases/Release/Article/2376629/dod-issues-new-data-strategy/.

DoDD 8140.01. 2020. *DoD Directive 8140.01, Cyberspace Workforce Management.* Washington, D.C.: Deputy Secretary of Defense. https://www.esd.whs.mil/Portals/54/Documents/DD/issuances/dodd/814001p.pdf.

DoDI 5000.87. 2020. "Operation of the Software Acquisition Pathway ." Office of the Under Secretary of Defense for Acquisition and Sustainment . https://www.esd.whs.mil/Portals/54/Documents/DD/issuances/dodi/500087p.PDF.

DoDI 8140.02. 2021. *DoD Instruction (DoDI) 8140.02, Identification, Tracking, and Reporting of Cyberspace Workforce Requirements.* Department of Defense (DoD) Chief Information Officer . https://www.esd.whs.mil/Portals/54/Documents/DD/issuances/dodi/814002p.PDF.

Dredd. 2024. *Dredd-HTTP API Testing Framework.* Accessed June 21, 2024. https://dredd.org/en/latest/.

EPA. 2023. "Learn about Data Standards." *U.S. Environmental Protection Agency (EPA).* June. Accessed October 23, 2023. https://www.epa.gov/data-standards/learn-about-data-standards#:~:text=What%20are%20data%20standards%3F,use%2C%20and%20management%20of%20data.

F5 Glossary. n.d. *API Gateway.* Accessed June 21, 2024. https://www.f5.com/glossary/api-gateway.

Firesmith, Donald. 2015. "A Taxonomy of Testing: What-Based and When-Based Testing Types." *SEI Blog.* insights.sei.cmu.edu/blog/a-taxonomy-of-testing-what-based-and-when-based-testing-types.

GitHub. 2024. *Microsoft/Restler-Fuzzer.* Accessed June 21, 2024. https://github.com/microsoft/restler-fuzzer.

Hoehn, John R. 2022. *Joint All-Domain Command and Control (JADC2).* IF11493, Congressional Research Service (CRS). https://us-east-1-02900067-inspect.menlosecurity.com/safeview-fileserv/tc_download/d4a9166ec39f3f84b1592e54ac08c10706a8025adf7cfa5d118462b3395e2b74/?&cid=N17BBEED83A26_&rid=ed191390276891c293ed9aa049739f.

Humble, Jez, and David Farley. 2015. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation.* Boston: Addison-Wesley.

IBM API. 2023. "What Is an API?" *IBM.* Accessed October 23, 2023. https://www.ibm.com/topics/api.

IBM Qiskit. 2024. *Qiskit Runtime - Create a Qiskit Runtime instance to run quantum programs, on Quantum Systems.* IBM. https://cloud.ibm.com/catalog/services/qiskit-runtime.

IEEE 2675-2021. 2021. *IEEE 2675-2021: IEEE Standard for DevOps: Building Reliable and Secure Systems Including Application Build, Package, and Deployment.* C/S2ESC Software and Systems Engineering Standards Committee, IEEE Computer Society. https://standards.ieee.org/ieee/2675/6830/ .

IEEE Computer Society. 2023. "Software Engineering Body of Knowledge Version 3." *IEEE Computer Society*, July 19. www.computer.org/education/bodies-of-knowledge/software-engineering/v3.

ISO 8601. 2019. *ISO 8601: Date and Time Format.* Geneva: International Organization for Standardization (ISO). https://www.iso.org/iso-8601-date-and-time-format.html.

ISO/IEC 7498-1. 1994. *ISO/IEC 7498-1:1994. Information Technology: Open Systems Interconnection Basic Reference Model.* Reviewed and confirmed in 2000, Geneva: International Organization for Standardization (ISO). https://www.iso.org/standard/20269.html.

Kim, Myeongsoo, et al. 2023. "Enhancing Rest API Testing with NLP Techniques." *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis.* https://doi.org/10.1145/3597926.3598131.

Kumar, Divya, and K. K. Mishra. 2016. "The impacts of test automation on software's cost, quality and time to market." *Procedia Computer Science* 79: 8-15. https://doi.org/10.1016/j.procs.2016.03.003.

Kutztown. 2014. *Invariants, Preconditions, and Postconditions.* Kutztown, Pa.: Kutztown University Department of Computer Science. Accessed July 2024. https://www.kutztown.edu/Departments-Offices/A-F/ComputerScienceInformationTechnology/Documents/Student%20Resources/DocumentationInvariants.pdf.

Leung, Adrian, and Chris Mitchell. 2023. "Service Discovery Threat Model for Ad Hoc Networks." https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=a400c6ac8890e523d3ee3cd17074ad9db0799085.

McKinsey & Company. 2023. "What Is Psychological Safety?" *McKinsey & Company.* July 17. Accessed October 24, 2023. https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-psychological-safety.

Mead, Nancy R., and Carol C. Woody. 2017. *Cyber Security Engineering: A Practical Approach for Systems and Software Assurance.* Boston: Addison-Wesley.

Meyer, Bertrand. 1997. *Object-Oriented Software Construction.* 2nd Edition. Prentice Hall.

NATO. 2023. *A Data Centric Architecture for the Alliance.* Working paper AC/322 (CP/1)WP(2023)0021 (INV), Consultation, Command and Control Board (C3B) Communications and Information Services Capability Panel (CIS CAP), North Atlantic Treaty Organization (NATO).

Nguyen, Cuong, Huy Bui, Vu Nguyen, and Tien Nguyen. 2023. "An Approach to Generating API Test Scripts Using GPT." *Proceedings of the 12th International Symposium on Information and Communication Technology (SOICT '23).* Association for Computing Machinery.

NIST SP 1800-21. 2020. "NIST Special Publication (SP) Mobile Device Security: Corporate Owned Personally-Enabled (COPE)." *NIST Computer Security Resource Center.* Edited by Joshua, et al. Franklin. National Institute of Standards and Technology (NIST). September. Accessed October 1, 2023. https://csrc.nist.gov/pubs/sp/1800/21/final.

NIST SP 800-204. 2019. *Security Strategies for Microservices-Based Application Systems.* National Institute of Standards and Technology (NIST), Department of Commerce. https://csrc.nist.gov/pubs/sp/800/204/final.

NIST SP 800-207. 2020. *Zero Trust Architecture.* National Institute of Standards and Technology (NIST). https://csrc.nist.gov/pubs/sp/800/207/final.

OAuth 2.0. 2023. *OAuth 2.0.* Accessed October 23, 2023. https://oauth.net/2/.

OUSD(A&S). 2019. *DoD Digital Modernization Strategy: DoD Information Resource Management Strategic Plan FY19-23.* Washington, D.C.: Office of the Under Secretary of Defense for Acquisition and Sustainment (OUSD(A&S)). https://media.defense.gov/2019/Jul/12/2002156622/-1/-1/1/DOD-DIGITAL-MODERNIZATION-STRATEGY-2019.PDF.

OUSD(A&S). 2022. *Guidance for Programs to Create API Strategy.* Washington, D.C.: Office of the Under Secretary of Defense for Acquisition and Sustainment (OUSD(A&S)). https://aaf.dau.edu/storage/2023/05/Program-API-Strategy-Template_22May2023_v2.docx.

OWASP. 2024. *Fuzzing.* Accessed March 8, 2024. owasp.org/www-community/Fuzzing.

Sanders, Geoffrey. 2021. *Integrating Zero Trust and DevSecOps.* Carnegie Melon University Software Engineering Institute . https://apps.dtic.mil/sti/trecms/pdf/AD1145432.pdf.

Sato, Daniel. 2014. "Canary Release." *martinFowler.com.* June 25. Accessed October 23, 2023. https://martinfowler.com/bliki/CanaryRelease.html.

Searls, Justin. 2021. ""People love debating . . . ."." *X.* May 14. https://twitter.com/searls/status/1393385209089990659.

Seemann, Mark. 2012. *TDD Test Suites Should Run in 10 Seconds or Less.* May 24. blog.ploeh.dk/2012/05/24/TDDtestsuitesshouldrunin10secondsorless/.

SEI . 2022b. "DevSecOps Platform Independent Model (PIM)." *Carnegie Mellon University (CMU) Software Engineering Institute (SEI).* May. Accessed June 21, 2024. https://www.sei.cmu.edu/our-work/projects/display.cfm?customel_datapageid_4050=344852.

SEI. 2022a. "Unlock the Potential of DevSecOps in Regulated and Constrained Environments." *Carnegie Mellon University Software Engineering Institute (SEI).* May. Accessed June 18, 2024. https://www.sei.cmu.edu/our-work/projects/display.cfm?customel_datapageid_4050=344852.

Sherman, Mark. 2024. *Using ChatGPT to Analyze Your Code? Not So Fast.* Software Engineering Institute (SEI), Carnegie Melon University. https://insights.sei.cmu.edu/blog/using-chatgpt-to-analyze-your-code-not-so-fast/.

Sindall, Gemma. 2020. "An API Governance Model for Great APIs." *digitalML.* August 28. Accessed October 23, 2023. https://www.digitalml.com/api-governance-model/.

—. 2023. "What Is API Governance? 9 Best Practices for API Governance Success." *digitalML.* April 23. Accessed October 23, 2023. https://www.digitalml.com/api-governance-best-practices/.

Swagger. 2021. *OpenAPI Specification, Version 3.1.0 February 2021.* Swagger. SmartBear. February. https://swagger.io/specification/.

Villalba, Marcia. 2023. *Previewing Environments Using Containerized AWS Lambda Functions.* AWS Compute Blog. February 6. Accessed March 2024. https://aws.amazon.com/blogs/compute/previewing-environments-using-containerized-aws-lambda-functions/.

Williams, Laurie, et al. 2009. "On the effectiveness of UNIT test automation at Microsoft." *20th International Symposium on Software Reliability Engineering.* https://doi.org/10.1109/issre.2009.32.

Yallon, Erez, et al. 2023. *OWASP Top 10 API Security Risks – 2023.* Open Worldwide Application Security Project (OWASP). https://owasp.org/API-Security/editions/2023/en/0x11-t10/.

Application Programming Interface (API) Technical Guidance

Minimum Viable Capability Release (MVCR) 1, July 2024

Office of Systems Engineering and Architecture
Office of the Under Secretary of Defense for Research and Engineering
3030 Defense Pentagon
Washington, DC 20301
osd-sea@mail.mil
https://www.cto.mil/sea