



OHIP Secure API Development Standards Baseline

Version 3.2 - June 2026

Overarching Guiding Principles for API Security

Application programming interface (API)s are mechanisms that enable two software components to communicate with each other using a common language. From a security viewpoint, it does not matter if the communication is happening externally, such as using Secure File Transfer Protocol (SFTP) to transfer a file over the internet; or internally, like when two programs within an organization that need to share information.

The principles and standards herein are aligned with the guiding principles and standards in NIST SP 800-228 where:

- API Security controls should cover all APIs, regardless of whether they are exposed to customers/partners or used internally within the enterprise to support the business functions of the externally exposed API services.
- There is no distinction between an “internal” and “external” caller (client requests) regarding the security controls that are applied to it.
- DevSecOps controls should span the entire iterative API life cycle including:
 - o Pre-runtime protections that need to be applied when designing and building APIs.
 - o Runtime protections that need to be applied to every API request and response that an infrastructure serves, not just at the perimeter.

Applicability

This document is a summary of specific security principles and underlying implementation standards in the Department’s API Security Technical Guidance document. The API Security Technical Guidance document contains detailed and heavily referenced implementation standards. The principles and implementation standards are based on multiple security standards definition organizations (SDOs), with each SDO contributing their separate domain expertise in API security to these standards. The SDO references include those from NIST, IETF, OpenID Foundation, and OWASP.

While the summaries of implementation standards listed here are very general, the detailed implementation standards in the technical guidance are specific in their requirements to meet those high-level implementation standards. The reader is referred to the technical guidance document Version 3 for detailed guidance, implementation specifics, and supporting references.

A number of the implementation standards herein are focused on the OAuth 2.0 RESTful API paradigm to support ONC and CMS regulatory use cases. OAuth Profile extensions, like HL7 SMART, and related CMS/ONC regulations aside, these specific control standards are directly applicable to any standards-based REST-based API. All of the guiding principles and remaining



implementation standards are generalizable to API security for APIs that are exposed via other exchange standards such as Simple Object Access Protocol (SOAP) or Remote Procedure Calls (gRPC).

API Concepts

Externally exposed api's hosted by an api gateway have endpoints that map to many different services. These are referred to as facade api's, because they present a single facade to an outside (external) caller (client application) over (potentially many) different internal service api's. Internal services communicate with each other via other internal api's in order to directly or indirectly support the business functions of the facade api (see NIST SP 800-228).

Regulatory Background and Rationale for API Security

As a CMS regulated payer and HIPAA Covered Entity (CE), API Security poses unique challenges for the Department.

For example, as a CE and regulated payer under CMS-9115-F, the Department is required to allow its members to designate a third-party client application of their choice to access their MCD via the Department's Patient Access API. Third-party client applications and their developers, on the other hand, are not regulated under HIPAA.

At the same time, as a HIPAA CE, the Department is required to ensure the security and privacy of protected data while it is the state's possession and while in transit to a client application. The Department is also responsible for ensuring the operational integrity, security and privacy of the systems supporting that API.

Other Sensitive / Confidential Data regulatory and policy compliance requirements expand the risk surface to the Department beyond Medicaid Confidential Data (MCD) and HIPAA (e.g., FTI, SSA, PCI). Other Sensitive Data exposure / misuse / attribution risks include intellectual property and administrative information that requires provenance / attribution to an authoritative source.

Multiple factors further add to both the complexity and challenges of API security assurance in this environment. These include:

- Increasingly, sophisticated attackers have moved up the stack from low-level exploits and DoS attacks to application-level attacks that leverage the APIs that systems use to functionⁱ.
- OAuth 2.0, the foundational REST-based client authentication and authorization standard for APIs, was developed and published by the IETF over 12 years ago. Recently, the IETF has published a new standard for Best Current Practice for OAuth 2.0 Security (RFC9700)ⁱⁱ. The RFC9700 security standard updates the foundational standards and notes that the updates are required because:
 - o OAuth is currently being used in environments with higher security requirements than considered initially, such as Open Banking, eHealth, or eGovernment. These use cases call for stricter guidelines and additional protection.

- o The initial assumption of the standard was that there is a static relationship between client, authorization servers, and resource servers. This is not the case today.
- o OAuth implementations are currently being attacked through known implementation weaknesses and patterns that are considered insecure.
- o The current state of OAuth use presents new challenges with respect to security. They demonstrate a need for more specific recommendations, easier to implement mitigations, and more defense in depth.
- National security standards organizations (NIST, IETF, OWASP, OpenID Foundation) have identified a broad range of client application vulnerabilities that can pose risks to APIs. Further, a large portion of client application implementations do not properly implement security controlsⁱⁱⁱ. For example, the recent CMS Final Rule (CMS-0057-F) explicitly lists examples of client application vulnerabilities that present unacceptable risk to the Patient Access API^{iv}.
- In addition, compliance with CMS Final Rule CMS-0057-F and the standards promulgated by ONC HTI-1 will require APIs to support backend service clients in addition to direct single user data requests. Here, an analogy to B2B exchanges, external backend service clients will have bulk access to multiple patients' data at a system level access scope.

Purpose

This document provides an audience-appropriate summary of definitions, standards, citations and justifications for baseline API security standards. They are established to meet state and federal requirements and industry best practices for protecting the security and privacy of Sensitive / Confidential data (defined below) associated with, or in support of, the New York State Medicaid program.

The standards are based on national standards and best practices. They provide objective, verifiable criteria for assessing risk and identifying countermeasures that directly address, or address root causes of risks to an API.

The standards within this document are intended to act as a foundation for guiding API procurement, development, implementation and risk assessment.

This document is also intended to support identification and prevention / mitigation of API vulnerabilities that present the potential for unacceptable risk for the following technical and business impacts^{iv} on the Department. These include:

Technical Impacts:

- Unauthorized access to / release of Sensitive / Confidential Data while in possession of the Department's API.
- Unauthorized access to / release of Sensitive / Confidential Data while in transit between an external client and the Department's API.
- Degradation or exploitation of the operational integrity or security of the Department's API or its systems.

Business Impacts:

- Non-compliance with state and federal privacy and security regulations.
- Loss of reputation. Financial loss.



Document Description and Overview

This document is a summary of specific guiding principles and underlying attribute standards in the Department's API Security and Privacy Technical Guidance document. The API Security and Privacy Technical Guidance document contains detailed and heavily referenced implementation standards for each Attribute within a Guiding Principle. The detailed implementation standards for each attribute are based on a composite of the standards and best practices listed in the reference section below.

While the summary attribute standards listed here are very general, the detailed implementation standards in the technical guidance document are specific in their requirements to meet those high-level attribute standards. Attestations to the Department's API Security and Privacy Standards are based on the implementation standards within the Technical Guidance Document. The reader is referred to the Technical Guidance Document for detailed guidance, implementation specifics, and supporting references.

Scope

These standards apply to internal and external APIs and internal microservices used to access, store, transmit, or process Sensitive / Confidential data of any type. This includes applications and systems that use or depend on internal or external APIs and internal microservices to support their functionality. (e.g., Artificial Intelligence systems, applications and models)

The term "Sensitive / Confidential Data" includes, but is not limited to:

- Medicaid Confidential Data(MCD),
- PHI / PII,
- SAMHSA / SUDS / Mental Health
- Reproductive Health
- Health of Minors
- FTI, SSA, PCI
- Information that, if exposed, could present a threat to system integrity / availability, Security / Privacy, or support adversarial reconnaissance or unauthorized access to protected resources (e.g., access tokens, system capabilities and guardrails). Information that essential to critical decision support for members, providers or state administrative staff.
- Intellectual property (e.g., source code, proprietary algorithms)
- Other program administrative information that requires provenance / attribution to an authoritative source (e.g. policies, regulations)

Use of Sensitive / Confidential Data includes, but not limited to, use of this data in any direct or indirect manner for the purpose of:

- Application development, code review, code documentation, code transformation or debugging
- Decision support
- Predictive clinical or administrative functions
- Payment / reimbursement
- Benefit assess / determination



- Eligibility / enrollment
- Payment integrity
- Adjudication of claims
- Prior authorization
- Member or Practitioner health / care access, management or provision
- Member or practitioner help desk or advisory functionality of any type
- Beneficiary / provider communications of any type
- Policy or regulatory development or distribution / interpretation of same
- Used in any way to support AI-based systems (e.g., Model / agent training, Augmentation / grounding, RLHF / data labeling, inference, etc.)

The API standards to be attested herein apply to all applications that are used to access, store, transmit, interpret, process Sensitive / Confidential Data for the purposes as described above.

The term API includes, but is not limited to:

- External and internal endpoints
- Microservices
- Micro APIs
- Service meshes
- API gateways
- Internal and external service-to-service or tool calls
- Any internal infrastructure / services that orchestrates, manages, authenticates or registers the above.

The applicability of any specific standard herein will depend on the use case of an application that leverages API technology. This would include considerations such as level risk related to confidentiality sensitivity or decision criticality classification of the application, its functionality, architecture, environment, exposure and complexity. The Department reserves the right to define what the applicable attestation criteria are based on the above criteria.

The entities covered within this scope includes, but is not limited to, APIs as defined above:

- Developed or used by vendors or contractors on behalf of DOH.
- COTS and SaaS offerings that leverage APIs of any kind.
- Developed or used by a data consuming entity for use with Medicaid or member data.
- Developed or used by OHIP.
- Developed or used by the Office of Information Technology Services (ITS) or other NYS agencies.

Regulations

APIs that fall within the Scope statements above shall adhere to the following regulations:

- Health Information Portability and Accountability Act (HIPAA).
- Center for Medicare & Medicaid Services (CMS) Interoperability and Patient Access final rule (CMS-9115-F)^v.



- Center for Medicare & Medicaid Services (CMS) Advancing Interoperability and Improving Prior Authorization Processes (CMS-0057-F)^{vi}
- Office of the National Coordinator for Health Information Technology (ONC). Health Data, Technology, and Interoperability: Certification Program Updates, Algorithm Transparency, and Information Sharing (HTI-1)^{vii}
- 42 CFR 431.60 Beneficiary access to and exchange of data^{viii}.
- 45 CFR 170.215 Application Programming Interface Standards as updated by ONC HTI-1^{ix}.
- Other standards under 45 CFR 170 and 171 as applicable to API Security and Interoperability standards.
- All New York State ITS policies and standards^x.

API Security Standards and Best Practice References Used and Cited in Developing the Document

- NIST 800-228 Guidelines for API Protection for Cloud-Native Systems^{xi}
- CMS (CMS-9115-F) mandated HL7 SMART Implementation Guide Version 1 (published in 2018)^{xii}, the foundational security standards it references and those referenced by them:
 - IETF RFC6749 The OAuth 2.0 Authorization Framework (2012)^{xiii}
 - IETF RFC6819 OAuth 2.0 Threat Model and Security Considerations (2013)^{xiv}
 - IETF RFC7636 Proof Key for Code Exchange by OAuth Public Clients^{xv} (2015) (referenced by RFC8252)
 - IETF RFC8252 OAuth 2.0 for Native Apps^{xvi} (2017)
- ONC mandated HL7 SMART Implementation Guide Version 2^{xvii} (replaces SMART Version 1 effective January 2026). (n.b., CMS has endorsed and recommended use of SMART Version 2 for backend services^{xviii})
- CMS (CMS-9115-F) mandated OpenID Foundation Connect Core 1.0 (OIDC), incorporating errata set 1^{xix} as part of the ONC API standards at 45 CFR 170.215.
- Standards and recommendations in the IETF Best Current Practice for OAuth 2.0 Security (RFC9700)^{xx}, along with the supporting RFCs it references.
- NIST 800-204 Series, Security Strategies for Microservices-based Application Systems^{xxi}.
- OWASP (web) Application Security Verification Standard 4.0.3 (ASVS)^{xxii}.
- OpenID Foundation Financial-grade API (FAPI) Security Profiles^{xxiii}

Companion Documents

This document is accompanied by five companion documents.

- OHIP Secure Application Programming Interface (API) Baseline Technical Guidance
- OHIP Application Programming Interface (API) Security Implementation Standards Baseline Attestation
- OHIP Artificial Intelligence (AI) Security and Privacy Baseline



- OHIP Artificial Intelligence (AI) Security and Privacy Implementation Standards Baseline Technical Guidance
- OHIP Artificial Intelligence (AI) Security and Privacy Implementation Standards Baseline Attestation

IMPORTANT NOTES FOR AI SYSTEMS:

- Given the dependency of AI Systems on internal and external API's as well as internal microservices infrastructure, the OHIP API Security Technical Implementation Standards Baseline must be attested to as a prerequisite to OHIP AI Security and Privacy Implementation Standards Baseline Attestation.
- Attestation to the OHIP AI Security and Privacy Baseline is based on the Implementation standards contained within the OHIP AI Security and Privacy Standards Technical Guidance Document.

API Security Best Practices and Standards

OHIP has developed the following set of ten principles for API Security. Each principle is accompanied by attribute standards that support the principle. The principles and implementation standards are based on authoritative national sources and established to meet the federal requirements of protecting Medicaid data. APIs and microservice implementations that fall within the Scope statements above, shall adhere to the following principles and attributes. Attestations to their implementation will be based on the implementation standards within the Technical Guidance Document, where deemed applicable.

OHIP has also developed a series of Overarching API Security Standards that govern all implementation requirements that underpin each of the attributes listed below. The specific implementation standards are provided in the Baseline Technical Guidance document. The Overarching API Security Implementation Standards are listed and described in Appendix 1 herein.

Instances where the standards reference the term “data” should be interpreted to mean Sensitive / Confidential Data.

The reader is advised to review the Overarching Implementation Standards as they cover the following important topics:

- The Department's expectations regarding existence of documentation and artifacts that are either implied or required by the attribute implementation standards.
- The Department's expectations regarding addressing the overall intent of a control attribute and / or specific implementation standard. (e.g., see examples regarding in interpretation of the term microservices)
- The Department's expectations regarding full and complete disclosure of all external third-party dependencies.

API-1 The API Service Implements the Technological Capabilities, Procedures, Protocols and User Interfaces as Needed to Allow the Department to Electronically Implement its Security Policies and Business Rules.

API 1-1 API technology provides an OpenID Connect Standard (OIDC) service in a manner that supports the Department's Identity and Access Management



(IAM) security policies and standards.

- API 1-2 The API Service includes an application access management capability that allows the Department to use its risk assessment methodology to specifically designate which client applications may access its API.
- API 1-3 The API Service ensures that users must periodically re-authenticate and re-consent their client application's access to the Department's API according to the Department's business rules.
- API 1-4 API technology and procedures support the Department's bearer token provision and expiration standards and policies using national standards organization's current best practices.
- API 1-5 API technology and bearer token exchange rules use national standards organization's current standards and best practices to support the Department's security standards, business rules and policies.

API-2 APIs Prevent Access to User-related Authentication and Authorization Credentials by Client Applications or Malicious Actors.

- API 2-1 [API standards](#) prohibit use of embedded user agents.
- API 2-2 [API technology](#) detects and defends against fake external agents.
- API 2-3 API technology detects when an application user authentication and / or authorization interface is embedded in a client application.
- API 2-4 API technology defends against malicious application or malware that hijacks user credentials as they are being entered.
- API 2-5 API standards and technology for implementing security strategies to prevent credential abuse, credential stuffing and other brute-force attacks.

API-3 Protecting Confidentiality and Integrity of Identity and Access Credentials Issued by the API to Client Applications.

This also applies to [internal services](#) used to support the API functions, such as [microservices](#) used by an [api gateway](#).

The API implements procedures and enforces technology standards for the issuance and protection of access credentials granted to external client applications or used by internal API services in the following ways:

- API 3-1 API client application registration standards apply security restrictions based on the type of client application registering to use the API.
- API 3-2 API client application registration requirements and access standards ensure that an application can protect client access credentials (e.g., client identity credentials, authorization code, access token, and refresh token) issued to it by the API on behalf of the Department.
- API 3-3 API standards and procedures for management of access credentials issued to an external client applications or internal services are in place.



- API 3-4 API technology enforces current best practice standards for protection of authorization codes from leakage, theft and replay.
- API 3-5 API technology enforces current best practice standards for protection against access and refresh token leakage, theft and replay.
- API 3-6 API technology implements, enforces, or supports current best practice standards to protect against threats to leakage or exfiltration within access credential flows between client applications and API, or internal services used by the API.
- API 3-7 The [API architecture](#) framework implements security strategies for maintaining session integrity and authenticity between internal API services, and where applicable between external client applications and the API.

API-4 Ensuring That the Principles of Least Privilege and Minimum Necessary are Applied to Client Application Access Rights and Internal Services Used to Support the API Functions.

The API implements procedures and enforces technology standards that apply the principles of minimum necessary and least privilege to external client applications and API internal services

- API 4-1 API technology enforces current best practice standards that ensure the principles of least privilege / minimum necessary are applied to access and refresh tokens issued to a client application.
- API 4-2 API technology enforces standards that protect against client applications impersonating a [resource owner](#).
- API 4-3 API service architecture standards, configuration, and technology enforce principles of least privilege and minimum necessary.

API-5 API Architecture Implements Internal Service Authentication and Access Control (Authorization) Based on Best Practices and Standards.

A centralized API architecture implements procedures and enforces technology standards that ensure reliable identity- and access-rights enforcement between internal service clients and their target services. This is achieved by policy-based conveyance, provisioning, and enforcement of access to and between all internal service exchanges, and external client facing API services and resources. API architecture is:

- API 5-1 Configured to ensure integration with authentication and authorization frameworks.
- API 5-2 Implemented using secure, standards-based authentication and authorization methods. For internal services those as recommended by NIST Security Strategies for Microservices-based Application Systems (NIST 800-204) and NIST Guidelines for API Protection for Cloud-Native Systems (NIST 800-228).
- API 5-3 Implemented using access and token management standards and



technologies that protect against token leakage, theft and injection or replay and ensure authenticity of requesting services.

API-6 API Architecture Implements Strategies for Secure Communication.

The API architecture provides secure, encrypted, end-to-end communication and authentication between client applications and API services ([north-south traffic](#)), as well as between internal API services ([east-west traffic](#)) and:

- API 6-1 Enforces all internal and external flows to be securely encrypted end to end.
- API 6-2 Implements strategies for secure, trusted, and [mutually authenticated communication](#).¹
- API 6-3 Enforces strong client / service authentication using current best practice standards for external client applications and internal services.
- API 6-4 Ensures that the [service mesh](#) architecture protects the proxy used for firewalling and blocking network traffic where applicable.

API-7 API Resiliency and Availability

To maintain resiliency, availability, and integrity through standards, architecture, and technology, the API uses security strategies that:

- API 7-1 Implements counter measures against cascading failures.
- API 7-2 Implements load balancing technologies and related infrastructure.
- API 7-3 Mitigates against oversubscribed services and Denial-of-Service (DoS) attacks.
- API 7-4 Supports service registry configuration, integrity, and availability of registry services.
- API 7-5 Implements procedures that allow for rapid reproducible recovery and deployment of all API component configurations.
- API 7-6 Correctly defines mesh architecture configuration standards and parameters associated with integrity and availability where applicable.

API-8 API Security and Performance Monitoring

The API implements security strategies to monitor, analyze, and detect events affecting security and availability of the API infrastructure, services, and functions and:

- API 8-1 Implements comprehensive approaches to security and performance monitoring.
- API 8-2 Captures high quality and useful content from logging sources to effectively monitor and detect for security and performance events across the API

¹ mTLS – mutually authenticated Transport Layer Security (TLS)
February 2026



infrastructure.

- API 8-3 Implements standardized logging attributes and secure approaches to logging and log content acquisition and protection.
- API 8-4 Implements centralized monitoring, visualization, and analysis.
- API 8-5 Monitors for event detection based on comparison against predetermined and validated baselines.
- API 8-6 All external client and internal service requests and responses are monitored and validated according to the related api's request and response schema as developed according to current best DevSecOps practices as described in API 9 below.

API-9 API Secure Development Life Cycle and Application Release Methodology

The API integrates Secure Development Life Cycle (SDLC) and [DevSecOps](#) best practices to ensure that security principles are built into the API architecture. This includes development, implementation, configuration, deployment, and future releases of all API components. These components include external Facade API services and endpoints as well as all internal services and api's that support the external Facade API's services.

- API 9-1 The API architecture / application development standards, procedures, and tools are based on SDLC and DevSecOps best principles and practices.
- API 9-2 All api's, whether internal or external that directly or indirectly support the functionality of exposed API endpoints and services are:

- o centrally governed,
- o identified,
- o inventoried,
- o documented and
- o have detailed best of practice specifications (including schema definitions) and are lifecycle managed.

These requirements include all development, test, staging and production environments definitions) and are lifecycle managed.

- API 9-3 Business logic security is built in to protect against likely internal or external threats.
- API 9-4 Input validation and output encoding architecture are designed to prevent injection attacks.
- API 9-5 Effective error handling assures that the API application fails safely.
- API 9-6 Effective security controls for specific input validation of all parameters that transition from a lower to higher trust level are in place for Web service, [REST](#) and [HTTP](#).
- API 9-7 API application ensures secure memory validation and sanitization.



- API 9-8 API architecture development mitigates against malicious code and ensures application integrity.
- API 9-9 API configuration architecture standards, procedures and tools ensure security is built into configuration management, build and deploy, and third-party dependency assurance processes.
- API 9-10 API architecture standards, procedures and technology implement security strategies for application and version release.

API-10 Foundational API Security Architecture Protections and Best Practices

The API architecture, as well as its components such as microservices, resources, registries, and supportive applications:

- API 10-1 Implement well-defined security controls for isolation, compartmentalization, and segregation of critical components.
- API 10-2 Establish general data protection controls to ensure confidentiality, integrity, and availability of sensitive, protected information.
- API 10-3 Implement access controls that require adequate authentication, management and authorization of all services, clients, and users.
- API 10-4 Implement a strong [cryptographic architecture](#) to protect data assets, resources and services from unauthorized access or disclosure.
- API 10-5 Implement strong stored cryptography that meets all required federal and industry standards for secure encryption algorithms, random values, and key management solutions commensurate with the data's sensitivity.
- API 10-6 API publishes metadata to improve the overall security and efficacy of [OAuth](#) deployments.

Appendix 1 Overarching API Security Implementation Standards

Overarching Security Implementation Standards		
API Overarching 1- Implicit / Explicit Existence of Documentation and Artifacts	Background	Many implementation standards within the technical guidance document use / include phrases such as: <ul style="list-style-type: none"> • Procedures • Policies • Written standards, API service provider standards, API standards • Are defined and documented • Requirements clearly define • Checklists, security requirements, policy guidelines are documented • follows a defined, documented and standards-based • Establishes mandated standards • Specifications, definitions • Tests or testing • Logic flows, permissions, schema, allow lists, input / output validation, configuration(s)
	The Department's Expectations	The Department's expectation for the response to that standard is that: <ul style="list-style-type: none"> • The API service provider has existing mature, detailed and written documentation that covers the specifics of the implementation standard. • The documentation includes a version history that demonstrates the level of maturity. • That the API service provider will provide that documentation / artifact to the Department in a timely manner upon request.
API Overarching 2- Responding to the Overall Intent of a Control Attribute and / or Specific Implementation Standard	Background	Multiple implementation standards use phrases such as: microservices, internal services, service registries, internal apis, etc.
	The Department's Expectations	The Department's expectation for the response to that standard is that it addresses the overall intent of the control attribute or implementation standard. Examples: <ul style="list-style-type: none"> • Where the API service provider believes that their application does not use microservice-based architecture, the provider should treat the control attribute and / or implementation

		<p>standard(s) as applying to the internal communication services and endpoints that exist within and between all components (compartments) that support their overall API application.</p> <ul style="list-style-type: none"> • Responding to attribute controls dealing with a service registry. The provider's architecture may not use a microservice registry or bus in the strictest definition of the terms. In that case, the provider would describe how all the internal services and endpoints within and between components that support their overall API application are centrally orchestrated, managed / governed, tracked and documented.
API Overarching 3- Disclosure of Third-Party Dependencies	Background	<p>API application service providers often utilize external Third-Party services or software providers to provide or support components of their application's functionality.</p>
	The Department's Expectations	<p>The Department's expectations are that:</p> <ul style="list-style-type: none"> • The API Service provider will separately provide full disclosure of all Third-Party services, software or libraries used in their API application. • The Disclosures will include: <ul style="list-style-type: none"> o Name, website, contact information of the Third-Party and contractual relations, if any, to the API Service provider. o Detailed documentation on what services, functionality and role it plays in supporting the overall API service provider's application. o The criticality (low, medium, high) of the role that the Third-Party service or software plays in supporting the availability, integrity and security of the overall API application. o Disclosure as whether the Third-Party service or software either directly or indirectly controls, enables or influences access to protected resources (e.g. PHI). This includes sensitive information (e.g., access credentials) that could lead to harm to the availability, integrity or security of the overall API application.



		<ul style="list-style-type: none">• Department will, at its discretion, make a determination as to whether third-party service or software will be required to separately address and attest to the applicable controls and implementation standards within this document.<ul style="list-style-type: none">o An example of when this would be required when a Third-Party has separately developed the software used by the API Service for externally facing OAuth Authorization and Resource server functionalities.
--	--	---

Glossary and Definitions

Security Standards Definitions

- **Application Programming Interface (API)** – Within the context of this document the term API (uppercase) refers to all externally facing and internal components that in total support the business function that the API is required support. Those components include externally facing endpoints and services as well as all internal api's, services, software, registries, data repositories and systems that directly or indirectly support that business function. The business functionality includes
 - Enables the [API Owner](#) to exchange data and access credentials securely, reliably, successfully, and accurately with an external client application on behalf of a user or backend process.
 - Ensures the operational integrity and resiliency of the API during appropriate use and threat-based behavior.
 - Ensures that users and external client application access to data exchange activities is rightfully attributable to scope allowed for the user or the backend process while that data is in possession of the API Owner.
 - Ensures that rights, responsibilities, and obligations of the API Owner adhere to HIPAA regulations.

An example of the use of the term “API” would be CMS mandated ‘Patient Access API’

- **API Owner** – a HIPAA covered entity, an entity who has a business associate agreement (BAA) with the HIPAA covered entity, a contractor of that covered entity, or a ‘covered data consumer’ of the HIPAA covered entity who wishes to make the covered entity’s data available through a separate API.
- **Data or Resource Owner and User** – see [OAuth Role](#) and other related definitions below.
- **API Standards** – organizational/enterprise policies, procedures, security standards, security best practices, and interoperability standards that:
 - o Meet federal and state standards.
 - o Ensure the above definition of API data exchange is met.
 - o Are consistently and verifiably implemented across all the API Owner’s APIs.
 - o Ensure secure implementation, maintenance, development, configuration, and management best practices.
 - o Are used to meet the attestation requirements for this standard.
- **API Technology** – API Technology is defined as the entirety of all systems, software, communications that manage, enable, secure, authorize, identify, and provide data that meets the definition of the API data exchange as described above.
- **API Architecture** – all API Standards and Technology in use to support API data exchange.

General Definitions

- **Protected Health Information (PHI)**

PHI includes all ‘individually identifiable health information’ held or transmitted by a covered entity or its business associate, in any form or media, whether electronic, paper,



or oral. Individually identifiable health information is information, including demographic data, that relates to:

- o the individual's past, present or future physical or mental health or condition,
- o the provision of health care to the individual, or
- o the past, present, or future payment for the provision of health care to the individual,
- o and that identifies the individual or for which there is a reasonable basis to believe it can be used to identify the individual. Individually identifiable health information includes many common identifiers (e.g., name, address, birth date, Social Security Number).

- **Personally Identifiable Information (PII)**

PII is any representation of information that permits the identity of an individual to whom the information applies to be reasonably inferred by either direct or indirect means.

Further, PII is defined as information:

- o that directly identifies an individual (e.g., name, address, social security number or other identifying number or code, telephone number, email address, etc.) or
- o by which an agency intends to identify specific individuals in conjunction with other data elements, i.e., indirect identification. (These data elements may include a combination of gender, race, birth date, geographic indicator, and other descriptors).

Additionally, information permitting the physical or online contacting of a specific individual is the same as personally identifiable information. This information can be maintained in either paper, electronic or other media.

- **Medicaid Confidential Data (MCD)**

MCD includes, but is not limited to:

- o Names and addresses of Medicaid applicants/recipients.
- o Medical services provided.
- o Social and economic conditions or circumstances.
- o Department of Health evaluation of personal information.
- o Medical data, including diagnosis and history of disease or disability.
- o Income, eligibility information, amount of Medicaid payment.
- o Information regarding the identification of third parties, and/or information the Department deems to be part of the operations of the Medicaid program that is of a sensitive nature, such as cost of services provided, and has repercussions if released.

- **api (lower case)**

An api (application programming interface) is a mechanism that enables an application or service to access a resource within another application or service. An api can be externally facing (e.g., an externally facing façade api or internal service api.) The application or service making the request is called the client (or caller).

The use of the lower case for the acronym 'api' refers to the actual physical implementation of a façade or internal service api.

- **Client Application**

The term client application, or client, refers to a caller of a service or endpoint. External client refers to an external application that makes calls against an externally facing api. Internal client or services refers to internal services that make calls that directly or indirectly support the externally facing service(s) that the external client is making requests

against.

- **Access credential flows**

Access credentials include all User, OAuth, OIDC and internal service codes or tokens issued by the Department or its internal or external API endpoints that convey identity, authorization, or access privileges. Access credential flows are the exchange processes that allow those access credentials to be requested from, issued by, validated, and presented to Department internal or external API endpoints. An example of access credential flows for OAuth Clients can be seen in the [supporting figure](#).

- **Internal Services**

Internal services are defined as all microservices or backend services that are directly or indirectly in a microservice architecture, whereby each microservice communicates with other services through simple interfaces to solve business problems. To serve a single user or external client request, a microservices-based application can call on many internal services to compose its response to meet its unique business requirement. In this paradigm, a third-party client request results in internal services acting like 'internal clients' themselves, calling other internal services to meet the calling service's business requirements.

As pointed out in NIST 800-204, the calling client and responding services must have specific authentication and identity management protocols as well as standards and privileges.

Foundational Technical Standards and Definitions

- **Representational State Transfer (REST)**

REST architectures are characterized by requests and responses for bidirectional communication between a client and a server. In this type of communication, the client requests resources from the server, which sends responses back to the client. The REST client-server paradigm operates with stateless, cacheable communication between clients and servers. It is a software architectural style for creating distributed applications that can communicate via HTTP. The World Wide Web may be regarded as a REST-based architecture.

The [HL7 FHIR](#) standard was designed and developed to interoperate over the World Wide Web RESTful Architecture.

- **Hypertext Transfer Protocol (HTTP)**

HTTP follows the REST architecture style. A web browser, for example, may be the client whereas a web server process, running on a computer hosting one or more websites may be the server. The client submits an HTTP request message. The client submits an HTTP request message to the server, the request contains a link to the resource (URL) and an operation (Method) requested for the resource such as GET, PUT, POST, or DELETE. The server returns resources to the client, or performs other 'Create, Read, Update, Delete' type operations on behalf of the client.

- **REST Resource**

REST architecture treats all server content as a resource. A REST resource *is an object that is identified by a URL (Uniform Resource Locator) provided by the REST server*. A resource in REST is like an object in Object Oriented Programming or is like an entity in a database. The REST server simply provides access to resources; a REST client accesses and modifies the resources. The most popular representations of resources are XML and JSON. FHIR Resources are no different and are typically in JSON format. Examples of FHIR resources include Patient, Medication, Procedure, Organization, Practitioner,



CarePlan, Problem, Encounter, and EoB.

- **RESTful api**

A RESTful api is an api that conforms to the design principles of the REST architectural style. REST api's have emerged as a common method for connecting components and applications in a microservices architecture. The HL7 FHIR standard was designed and developed to provide access to FHIR Health Resources using RESTful api's. RESTful api's conform to five key design principles: uniform interface (unique URI per resource), client- server decoupled, statelessness, cacheability, and layered system architecture.

- **OAuth 2.0 Definition (OAuth)**

OAuth 2.0, or 'OAuth', is an internet/web authorization standard published by the IETF in 2012 and is based on the RESTful api paradigm. It is the de facto industry standard for internet authorization and the IETF has published several updates and security best practices to accommodate its use in more sophisticated and secure applications.

The standard allows a user to grant a client application access to the user's (or to some individual whom the user has access rights to) protected resources, without necessarily revealing the user's long-term credentials – or even their identity.

To affect this level of abstraction, OAuth introduces an authorization layer that separates the role of the client from that of the user. The user consents to convey and delegate their access rights to protected resources (or to some individual whom the user has access rights to) to a third-party app, and in modern sense to the client application developer's data service.

In OAuth, the client application requests access to resources controlled (consented) by the user and hosted by the resource server; the client is issued a different set of credentials than those of the user. The client obtains an access token which is a string denoting the specific scope of data (resource) access that a client has, the lifetime it is good for, and other access attributes. The authorization server, with the approval of the user, issues access tokens to the client. Then, the client uses the access token to access the protected resources hosted by the resource server ([supporting figure](#)). Notably, the authorization server may also issue an optional 'refresh token' to the client to allow them to continually mint new access tokens when an access token expires.

- **OAuth 2.0 Role Definitions (Resource Owner, Resource Server, Authorization Server, Client)**

The OAuth standard defines several roles that are key to understanding how the authorization abstraction level conveys access and key privacy issues.

- o Resource Owner: The entity that grants/delegates access to protected resources to a client. Typically, the resource owner is likely to be the user, but not necessarily.
- o Resource Server: The server hosting the resource owner's protected resources. This is the API that the user wants to grant the client access to.
- o Client: The client application requesting access to protected resources on behalf of the user or Resource Owner.
- o Authorization Server: The server that authorizes the resource owner and issues access tokens after obtaining user authorization.

- **OAuth 2.0 Client and Application Definitions (Private Client, Public Client, Native app)**

The OAuth standard defines client and application types that are essential to understanding key security considerations. These are regarding unauthorized release of



protected resources while in possession of a resource server (HIPAA covered entity).

The important consideration is that the user, who is likely also the resource owner, is delegating all access to a third-party client application to their protected resources on the resource server. It is critical that the authorization server registers and issues client identity credentials to the client and that the authorization and resource servers trust them when presented by a client. In essence, the client has become the user, and the same client is reused by many users.

The original OAuth standard defines two types of clients: confidential and public clients. The type of client definition affects the type of authentication the apps can use to meet the required trust level.

- o Confidential clients can hold access credentials in a secure way without exposing them to unauthorized parties. They require a trusted backend server to store the secrets. Typical examples would include automated backend processes that download data in bulk from a resource server (B2B) and Backend for Frontend (BFF) services, where a single, isolated, managed client acts as the sole OAuth client interacting directly with the target API on behalf of many users. In either case, the Department's API access credentials are in possession within an organization's "protected perimeter", rather than an application on a user managed device.
- o Public clients cannot protect access credentials securely. A typical example would be a web browser-based application.

After updating the standard in 2017, IETF also defined a type of client application that is associated with being a public client. The update defined 'native apps' as being an application that a user installs on their device. In general, native applications are public clients. The user has control over what software is installed on their device; mobile apps are thus native apps. The 2017 update describes security requirements and other recommendations for native and mobile applications using OAuth 2.0. The update also defines how a public native application can be considered a confidential app. 2021 and 2022 IETF updates have imposed additional security requirements and recommendations for native/public applications.

Because confidential clients can hold secrets, client identity credentials are issued to them in one of two ways:

- o Symmetrically, using their client secret (HS256).
- o Asymmetrically, using a private key (RS256).

The most recent best practices are requiring asymmetric methods.

As public clients are unable to hold secrets, client identity credentials should not be issued to them. Indeed, recent updates to Financial-grade API Security profiles by the OpenID Foundation states that public clients should not be supported as OAuth clients directly accessing the target API. At the same time, when a public client interacts directly with a Backend for Frontend (BFF), which proxy's that client's access, as a confidential client, then that architectural pattern would support a public client's (indirect) access to the target API.

- **User Agent (Embedded and External)**

A web browser is a user-agent. An external user-agent would be the system browser provided with a mobile devices operating system (e.g., Safari for iOS); the user-agent is external to the client application. System browsers are the preferred external user-agent for user authentication or authorization as they provide the security of a separate process

for authentication and usability of a shared authentication session.

An embedded user-agent is a web browser user interface component that can be embedded in applications to render web pages. They are an alternative method for authorizing users through third-party apps. However, by definition, they are unsafe for use by third parties. The use of embedded user-agents in authorization/authentication requests should be prohibited in third-party apps because it allows the client to access the user's identity credentials.

- **OAuth Authorization Grant Code flow**

The original OAuth standard includes several approaches to granting an OAuth client access to protected resources on the resource server. More recent IETF OAuth best security practices require that the Authorization Grant Code flow approach be used. The [supporting figure](#) shows the authorization server issuing the client an authorization code after the user identifies themselves and conveys permission to the authorization server. The authorization code is provided back to the client using a pre-registered link (aka redirect URL). The client then exchanges that authorization code for an access token from the access server to request the user's protected resources from the resource server.

- **Authorization Grant Code flow with Proof Key for Code Exchange (PKCE)**

It is essential that the client can protect the access code while it is in the client's possession. It is also essential that the authorization server trust that the code is from the client that it was issued to. There are several ways in which the authorization code can be intercepted and re-played, or injected into, the authorization flow to the authorization server by malicious actors or apps. This issue is especially true for public clients (native mobile apps) which are unable to protect a client secret in the OAuth sense.

Proof Key for Code Exchange (PKCE), pronounced "pixie", is an OAuth extension that is designed to protect the authorization code flow. PKCE is based on proof of possession. It answers the question as to whether this authorization code is truly from the client it was issued to. Here, the client application gives unique proof to the authorization server that the 'authorization code' belongs to the client application for the authorization server to issue an access token. Proof of possession is therefore essential to trusting authorization codes from native apps which, by definition, cannot protect a client ID and secret.

The PKCE standard requires that both OAuth server and client support it. Here, the client sends a separate one-time 'code challenge' and 'challenge method' along with the authorization code request to the authorization server. Upon receipt of the authorization code from the authorization server, the client uses it to make the access token request by including a verification code. The authorization server then uses code challenge and challenge method to evaluate the verification code as to proof of possession. The authorization server returns an access token only if it can verify the verification code using the code challenge and challenge method.

PKCE has two main benefits: it protects the client from Cross Site Request Forgery attacks and the authorization server from authorization code injection attacks.

Recent IETF OAuth security best practices require PKCE for public clients (native, mobile apps) and strongly recommend it for confidential clients. The most recent IETF best practices also require it to be enforced for all clients accessing the API to avoid PKCE downgrade attacks.

- **OpenID Connect Core 1.0 (OIDC)**

OpenID Connect 1.0 is identity layer on top of the OAuth 2.0 protocol. It allows clients to verify the identity of the user based on the authentication performed by an authorization server, as well as to obtain basic profile information about the end user in an interoperable



and REST-like manner. OpenID Connect also allows clients to request and receive information about authenticated sessions, for example, whether 2FA was used to authenticate the user. It also allows the server, with the user's consent, to return information about the user, such as name, DOB, and so on. The exact user informational elements (or claims) returned to the client about the user is determined by the authorization server.

- **Health Level 7 (HL7)**

In addition to being the name of a [Standards Organization](#), HL7 also refers to a suite of international standards for exchange of clinical and administrative data between software applications used by various healthcare organizations and Health Information Exchanges (HIE). There are two versions of "HL7": Version 2 which is message based and uses a structure of pipe delimited records and Version 3 which is based on a Reference Information Model and XML.

- **HL7 Fast Healthcare Interoperability Resources (FHIR®)**

HL7 FHIR is an interoperability standard developed by the healthcare industry under the auspices of the Standards Organization, HL7. FHIR is intended to facilitate the exchange of healthcare and administrative information between healthcare providers, patients, caregivers, payers, public health researchers, and anyone else involved in the healthcare ecosystem. It is designed to leverage the standards and architectural styles used by the World Wide Web, such as REST.

The standard consists of two main parts: a content model in the form of resources, and a specification for the exchange of these resources in the form of real-time RESTful interfaces (e.g., apis) as well as messaging and documents. FHIR is based on an information model where resources have hierarchical relations and can be assembled to develop profiles that can support a wide array of HIE use cases. There are many FHIR resources, such as: Patient, Medication, Procedure, Organization, Practitioner, CarePlan, Problem, Encounter, EoB.

Version 4 of FHIR was adopted as a mandated standard by the Office of National Coordinator for Health IT for APIs used by all Certified Electronic Health Record Technology. The same version of FHIR was also adopted by CMS as the mandated standard for APIs providing beneficiary access to administrative claims data.

- **HL7 Substitutable Medical Applications, Reusable Technologies (SMART)**

The SMART Version 1.0.0 is an Application Launch Framework Implementation Guide to be used by client application developers that need to access protected FHIR health information resources using the OAuth 2.0 compliant authorization standard. In addition to the implementation requirements, the Guide also calls out and references multiple IETF OAuth standards and security practices. CMS mandates require regulated payers to implement RESTful api's that support this Guide.

Technology and Architecture Definitions

- **Microservices**

Microservices architecture, often shortened to microservices, refers to an architectural style for developing applications. Modern cloud-native applications are usually built as microservices. In a microservices architecture, business functions are packaged and deployed as services within containers and communicate with each other using API calls. This architecture approach allows a large application to be separated into smaller independent parts, with each part having its own realm of responsibility. Within a microservices architecture, each microservice is a single service built to accommodate an

application feature and handle discrete tasks.

- **North-South Traffic and East-West Traffic**

In classic data center paradigms, network traffic moves in two directions: north-south and east-west. North-south network traffic refers to traffic flowing south (into) and north (out of) the datacenter. East-west traffic, also known as lateral traffic, is the traffic from one service to another inside a network.

The api/microservice analog is that external client to-from a microservice is north-south traffic. Internal microservice-to-microservice exchange is east-west or lateral traffic.

Secure communication between external clients and services (north-south traffic) and between services (east-west traffic) is critical for the operation of a microservices-based application.

- **api Gateway**

An api Gateway is software that takes an application user's request (external client's request), routes it to one or more backend services, gathers the appropriate data, and delivers it to the user in a single, combined package. It also provides analytics, layers of threat protection, and other security for the application. An api Gateway is therefore north-south traffic focused (it may also have an east-west traffic focus – see NIST 800-204). Specifically, it sits between external clients and services. It acts as a reverse proxy, routing requests from external clients to services. The api Gateway provides a single-entry point for external clients accessing services behind it.

Specifically, the api gateway and its management environment:

- o Act as an abstraction layer to microservices/backend services by accepting external api calls and routing them to appropriate services or backend apis that it supports.
- o Orchestrate, authenticate, and manage internal service calls.
- o Verify credentials that are presented with external or internal requests.
- o Enforce usage quotas and rate limits.
- o Establish and manage registration, discovery, versioning, development, orchestration, telemetry, security, policy, performance, and privilege of each service and the services it interacts with.
- o Transform requests and responses as specified in policy statements.
- o Emit logs, metrics, and traces for monitoring, reporting, and troubleshooting.

- o **Service Mesh**

A service mesh is entirely focused on management and orchestration of internal service to service exchanges. It is therefore east-west traffic focused. Service meshes and api Gateways operate within different layers of a typical software architecture stack. The api Gateway is typically a public-facing infrastructure component, sitting between the edge of a network and the back end of an application—including the service mesh. A service mesh operates alongside each instance of a containerized microservice and abstracts the communication logic and security away from that microservice. This provides consistent tools to ensure managed, observable, and secure communication between individual services.

A service mesh automatically keeps track of the location and identity of each microservice, moving data between microservices. For example, Service A wants to share data with Service B. In this case, Service A simply channels that data to the service mesh implementation, which will then forward the data to Service B. The service mesh will eventually collect a response from Service B and will return it to Service A.

Current best practice approaches indicate that a service mesh should be used in



combination with an api Gateway.

- **Microservice Registries**

Microservice registries support two main functions: service discovery and service registry.

- o Service discovery is a core supportive function within a microservice architecture and an essential infrastructure service for api Gateway and service mesh architectures. In a microservices-based application, there are a substantial number of services, and many instances associated with each service with dynamically changing locations, calling for a robust service discovery mechanism. Discovery is the counterpart to registration from the client's viewpoint. When a client wants to access a service, it must find out where the service is located and other relevant information to perform the request. There are two approaches to discovery: client-side and server-side. Client-side discovery forces clients to query a discovery service before performing the actual requests. Server-side discovery makes the api Gateway handle the discovery of the right endpoints for a request. Best practices are seen as using a combination of the two discovery types. Service-side discovery can be used for providing access to all public APIs; client-side discovery can be used for internal service interactions. Microservice-specific security threats include those related to the service discovery mechanism.
- o Service registry is a database populated with information on how to dispatch requests to microservice instances. The registry keeps track of available instances and locations of each microservice. In addition, whenever an external client or internal service client needs to interact with another service, the client must discover information about the service and/or its endpoint location from the service registry. To be 'discovered' a service must be in the registry. The registry must also provide functionality to de-register a service; third-party service registration is normally used in the industry. In this case, there is a process or service that manages all other services known as the service manager. The service manager process polls or checks in some way which microservice instances are running or stopped and it automatically updates the service registry. It may also start and stop and detect service outages.

- **Cryptographic Architecture**

A cryptographic architecture consists of consistent policies, procedures, standards, and technology that ensure secure key management and storage, and strong encryption is built in to, and accessible by, all organizational applications. The technical implementation is typically a cryptographic subsystem that contains a cryptographic engine operating within a protected boundary that is accessible to applications via a controlled API.

- **Development, Security, and Operations (DevSecOps)**

DevSecOps is an approach to culture, automation, and platform design that integrates security as a shared responsibility throughout the entire IT lifecycle. It is the practice of integrating security testing at every stage of the software and system development process. It includes tools and processes that encourage collaboration between developers, security specialists, and operation teams to build software or systems that are both efficient and secure. DevSecOps brings cultural transformation that makes security a shared responsibility for everyone who is building the software or system. An Overview of of DevSecOps may be found in U.S. Department of Defense Enterprise DevSecOps Fundamentals. ^{xxiv}

- **Mutual Transport Layer Security (mTLS)**



mTLS is a mechanism for mutual authentication between services or between an external client application and the authorization service for an API. Also known as two-way authentication, it ensures that the parties at each end of a connection are who they claim to be. It is based on PKI, where both parties use X.509 digital certificates to authenticate each other. It operates on the premise of a Zero Trust security framework to verify devices, servers, and API connections, where trust of a client or API identity is not a given.

- **JSON Web Tokens (JWTs)**

JSON stands for JavaScript Object Notation. It is a way to store and transmit data. A JWT is a type of JSON element that is a compact, URL-safe means of representing data to be transferred between two parties.

Standards Organizations

- **Internet Engineering Task Force (IETF)** – founded in 1986, the IETF is the standards development organization for the Internet. It develops technical and engineering documents that influence the design, use, and management of the Internet. These documents include protocol standards, current best practices, and various kinds of informational documents.
- **Health Level 7 (HL7)** – an ANSI-accredited Standards Development Organization dedicated to providing a comprehensive framework and related standards for the exchange, integration, sharing and retrieval of electronic health information that supports clinical practice and the management, delivery, and evaluation of health services.

HL7 also refers to a suite of international standards for the exchange of clinical and administrative data between software applications used by various healthcare organizations and HIEs.

- **National Institute of Standards and Technology (NIST)** – develops cybersecurity standards, guidelines, best practices, and other resources to meet the needs of U.S. industry, federal agencies, and the broader public. Their activities range from producing specific information that organizations can put into practice immediately, to longer-term research that anticipates advances in technologies and future challenges.
- **Open Web Application Security Project® (OWASP)** – OWASP is a nonprofit foundation that works to improve the software security. It produces freely available articles, methodologies, documentation, tools, and technologies in the field of web application security.
- **OWASP Application Security Verification Standard (ASVS)** – provides a basis for testing web-application technical security controls and provides developers with a list of requirements for secure development.

The primary aim of ASVS is to normalize the range in the coverage and level of rigor available in the market when it comes to performing web-application security verification using a commercially workable open standard. The standard provides a basis for testing application technical security controls, as well as any technical security controls in the environment.



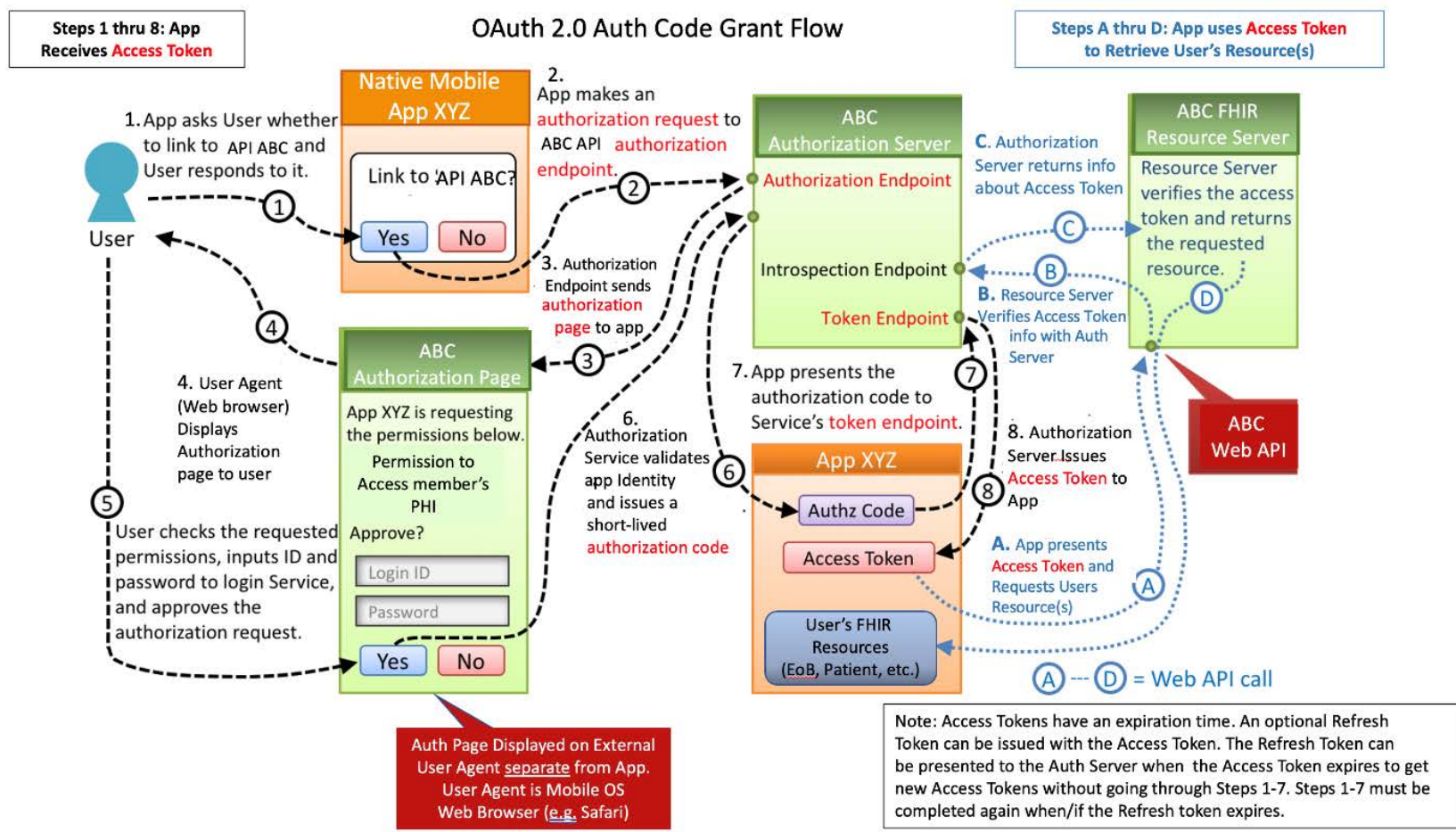
Versioning

Version History			
Version	Publication / Revision Date	Author(s)	Nature of Change
1.0	September 2023	J. Wilson, R. Wheeler, S. Dikeman, I. Gotham	Initial Draft
2.0	6/20/24	I. Gotham	Made corrections to better align API Security Baseline and Guidance documents. Updated references to regulations, standards and best practices to reflect new regulations (CMS-0057-F and ONC HTI-1) and most recent IETF Best Current OAuth Security Practices (BCP) and attacker models. Added Background and Rationale Section. Expanded Purpose Section. Added Version Table
3.0	6/26/25	I. Gotham	Revised background and rationale section. Added a new best practice section, now API-1, to include require meet to meet Department business rules. Revised various standards to accommodate changes to Version 3 of the detailed technology guidance document. Added new IETF RFC9700 standard for IETF Best Current OAuth Security Practices. Replaced 'third-party app' with 'client application' to include concepts of backend services and backend for frontend services, Added references for standards and best practices cited. Added and modified Definitions as appropriate. Added references to Financial-grade API (FAPI) Security Profiles.
3.0	10/15/25	I. Gotham	Replaced bullets for guiding principle attributes with numbering conventions that match the technical guidance document to allow. easier crosswalk to the implementation standards in the Technical Guidance Document. Harmonized wording and section conventions with the AI Baseline Security



			and Privacy document. <ul style="list-style-type: none">o Modified first 2 paragraphs of the Purpose section.o Modified the Scope Section.o Modified the Companion Documents Section.o Modified the first paragraph under the API Security Best Practices and Standards Section.o Added a Document Description and Overview Section.o Broke out Regulations and Standards Section into two sections.
3.1	2/4/26	Security and Privacy Bureau	Updated language in attribute API 9-9
3.1	5/28/26	Security and Privacy Bureau	Harmonized language regarding Sensitive / Confidential Data with AI Baseline Security and Privacy Documents.
3.2	June, 2026	R. Wheeler	Accepted all changes, removed comments, and added "Standards" to the title.

Supporting Figure



ⁱ Section 3 Recommended Controls for APIs
NIST SP 800-228 Guidelines for API Protection for Cloud-Native Systems
<https://csrc.nist.gov/pubs/sp/800/228/final>



-
- ii IETF Web Authorization Protocol Workgroup June 2024
<https://datatracker.ietf.org/doc/html/draft-ietf-oauth-security-topics>
- iii Section 1. Vetting the Security of Mobile Applications. NIST Special Publication 800-163 Revision 1 2019
IETF RFC 9700 Best Current Practice for OAuth 2.0 Security January 2025
<https://datatracker.ietf.org/doc/rfc9700/>
OWASP Mobile Application Security Testing Guide v1.5.0 2022
<https://github.com/OWASP/owasp-mastg/releases/tag/v1.5.0>
Vetting the Security of Mobile Applications. NIST Special Publication 800-163 Revision 1 2019
<https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-163r1.pdf>
OAuth 2.0 for Native Apps 2017
RFC 8252: OAuth 2.0 for Native Apps (rfc-editor.org)
OAuth 2.0 Threat Model and Security Considerations 2013
<https://datatracker.ietf.org/doc/html/rfc6819>
Section 4.10.3 IETF web authorization protocol workgroup. OAuth 2.0 Security Best Current Practice (BCP) February 2024
See also Section 4.9 in IETF BCP publications from 2021, 2022, 2023,
<https://datatracker.ietf.org/doc/html/draft-ietf-oauth-security-topics>
App Threat Report: The State of iOS App Security. Promon 2024. <https://promon.co/app-threat-report-ios-app-security/>
You Shall not Repackage! Demystifying Anti-Repackaging on Android. Computers and Security Volume 103, April 2021, 102181 <https://doi.org/10.1016/j.cose.2021.102181>
IETF RFC6819 OAuth 2.0 Threat Model and Security Considerations 2013
<https://datatracker.ietf.org/doc/html/rfc6819>
IETF web authorization protocol workgroup. OAuth 2.0 Security Best Current Practice 2024.
<https://datatracker.ietf.org/doc/html/draft-ietf-oauth-security-topics>
- iv Page 8778 Advancing Interoperability and Improving Prior Authorization Processes CMS-0057-F [2024-00895.pdf \(govinfo.gov\) February 8, 2024.](#)
- v CMS-9115F Interoperability and Patient Access.
<https://www.govinfo.gov/content/pkg/FR-2020-05-01/pdf/2020-05050.pdf>
- vi CMS-0057F Advancing Interoperability and Improving Prior Authorization
<https://www.govinfo.gov/content/pkg/FR-2024-02-08/pdf/2024-00895.pdf>
- vii HHS ONC HTI-1 Health Data, Technology, and Interoperability: Certification Program Updates, Algorithm Transparency, and Information Sharing.
<https://www.federalregister.gov/d/2023-28857>
- viii CMS 42 CFR 431.60(e) Beneficiary access to and exchange of data. Denial or discontinuation of access to the API
[https://www.ecfr.gov/current/title-42/chapter-IV/subchapter-C/part-431/subpart-B/section-431.60--p-431.60\(e\)](https://www.ecfr.gov/current/title-42/chapter-IV/subchapter-C/part-431/subpart-B/section-431.60--p-431.60(e))
- ix ONC 45 CFR 170.215 Application Programming Interface Standards
<https://www.ecfr.gov/current/title-45/subtitle-A/subchapter-D/part-170/subpart-B/section-170.215>
- x New York State ITS policies and standards.
<http://its.ny.gov/eiso/policies/security>
- xi NIST SP 800-228 Guidelines for API Protection for Cloud-Native Systems
<https://csrc.nist.gov/pubs/sp/800/228/final>
- xii HL7 SMART App Launch Version 1.
<https://hl7.org/fhir/smart-app-launch/1.0.0/>
- xiii IETF RFC6749 The OAuth 2.0 Authorization Framework
<https://datatracker.ietf.org/doc/html/rfc6749>
- xiv IETF RFC6819 OAuth 2.0 Threat Model and Security Considerations
<https://datatracker.ietf.org/doc/html/rfc6819>
- xv IETF RFC7636 Proof Key for Code Exchange by OAuth Public Clients
<https://datatracker.ietf.org/doc/html/rfc7636>
- xvi IETF RFC8252 OAuth 2.0 for Native Apps
<https://datatracker.ietf.org/doc/html/rfc8252>



^{xvii} HL7 SMART App Launch Version 2.0.0
<https://hl7.org/fhir/smart-app-launch/STU2/>

^{xviii} CMS Fact Sheet Interoperability and Prior Authorization Final Rule CMS-0057-F. See Recommended Implementation Guides
<https://www.cms.gov/files/document/fact-sheet-cms-interoperability-and-prior-authorization-final-rule-cms-0057-f.pdf>

^{xix} OpenID Connect Core 1.0 incorporating errata set 1
https://openid.net/specs/openid-connect-core-1_0-errata1.html

^{xx} IETF RFC 9700 Best Current Practice for OAuth 2.0 Security January 2025
<https://datatracker.ietf.org/doc/rfc9700/>

^{xxi} NIST 800-204 Security Strategies for Microservices-based Application Systems
<https://csrc.nist.gov/publications/detail/sp/800-204/final>
<https://csrc.nist.gov/publications/detail/sp/800-204/final>
NIST SP 800-204C Implementation of DevSecOps for a Microservices-based Application with Service Mesh.

<https://doi.org/10.6028/NIST.SP.800-204C>
NIST SP 800-204D Strategies for the Integration of Software Supply Chain Security in DevSecOps CI/CD Pipelines.
<https://doi.org/10.6028/NIST.SP.800-204D>

^{xxii} OWASP (web) Application Security Verification Standard 4.0.3 (ASVS)
<https://github.com/OWASP/ASVS/tree/v4.0.3/4.0>

^{xxiii} see

OpenID Foundation. 2022. Open Banking, Open Data and Financial-grade APIs. A Whitepaper for Open Banking and Open Data Ecosystem Participants Globally

https://openid.net/wordpress-content/uploads/2022/03/OIDF-Whitepaper_Open-Banking-Open-Data-and-Financial-Grade-APIs_2022-03-16.pdf

OpenID Foundation. 2022. Open Banking and Open Data: Ready to Cross Borders?

https://openid.net/wordpress-content/uploads/2022/07/OIDF-Whitepaper_Open-Banking-Open-Data_1st-Editors-Draft_2022-07-29.pdf

OpenID Foundation. 2022. Financial-grade API (FAPI) Profiles.

https://openid.net/wordpress-content/uploads/2022/08/OIDF_FAPI-Profiles-Comparisons_2022-07-27.pdf

OpenID Foundation. 2021. Overview of the FAPI Profiles.

https://openid.net/wordpress-content/uploads/2021/07/OIDF_FAPI_PROFILES_OVERVIEW.pdf

^{xxiii} OpenID Foundation. 2022. Financial-grade API Security Profile 1.0 - Part 1: Baseline

https://openid.net/specs/openid-financial-api-part-1-1_0.html

^{xxiv} OpenID Foundation. 2022. Financial-grade API Security Profile 1.0 - Part 2: Advanced

https://openid.net/specs/openid-financial-api-part-2-1_0.html

^{xxv} U.S. Department of Defense Enterprise DevSecOps Fundamentals. Version 2.5.

<https://dodcio.defense.gov/Portals/0/Documents/Library/DoD%20Enterprise%20DevSecOps%20Fundamentals%20v2.5.pdf>