Framework for Three-Party Authentication and Authorization of HTTP-Delivered Services in a Healthcare Setting

Matt Randall, Cerner Corporation

Audience

This document is intended for:

- Security specification authors
- Electronic Health Record (EHR) developers
- Application developers

Goal

This document is intended to complement, aid, and influence the development of open standards for EHR interoperability—specifically the HL7® FHIR® standard[1]—to ensure security risks are identified and mitigated in consistent manner, to maximize interoperability.

Executive Summary

To make off-the-shelf interoperability a reality in healthcare, security protocols are needed that standardize how applications can obtain the authorization to utilize EHR services on the behalf of a user. To further the development of secure, consistent, and interoperable implementations of EHR services and applications in this ecosystem, **this document provides the following:**

- An enumeration of the **use cases** (user stories), risks, and threats that must be considered when developing security specifications by specification authors.
- A **base framework** to be considered by security specification authors that addresses the use cases and identified risks/threats.
- A list of **functional and technical considerations** for software developers when implementing security services, EHR services, or consuming applications.
- A list of **requirements and best practices** that are recommended for both specification authors and software developers to agree upon to enable consistent, secure, and interoperable implementations to be developed.

[1] http://www.hl7.org/fhir/

Table of Contents

User Stories

Within the health-care environment, multiple actors may be involved with an individual's health record. When an application needs to access data in the EHR, it may need to interact (directly or indirectly) with one or more of these actors to obtain authorization to perform work.

The following terminology is used to describe actors in these stories:

- **Care Provider** – An individual who is utilizing a system as part of their duty in providing health care to other individuals.
- **Care Consumer** – An individual who is utilizing a system to access their personal healthcare information, or on behalf of an individual for whom they are legally responsible for.
- **Risk and Compliance Officer** – An individual who is responsible for ensuring that the policies, procedures, and controls (technical, functional) followed by an organization are sufficient to comply with government regulations and lie within an organization's tolerance for risk.
- **EHR Administrator** – An individual who configures an electronic health record in accordance with the organization's business requirements.
- **EHR Developer** – An individual responsible for the development of an electronic health record and associated services that it provides.

The following terminology is used to describe systems:

- **Client Application** – Software that provides new features by consuming web services exposed by the electronic health record.
- **Native Application** – Any client application in which the user does not interact via a web browser.
- **Web Application** – Functionality delivered via a web application that acts as a client application.
- **Shared Device** – A computing device with no mechanism for separating user access, shared by two or more individuals.
- **Resource Server** – A service that exposes information or functions within the electronic health record to client applications.
- **Authorization Server** – A system responsible for authorizing client applications to access a given resource server's resources.
- **Workstation** – An electronic computing device that provides access to the electronic healthcare record or associated systems.
- **Kiosk** – An electronic computing device that has a dedicated purpose in a care venue, such as providing self-service check-in for patients, or displaying information at a nursing station.

The following terms are used to describe state mechanisms in these systems:

- **Session** – A period of continuous access to a system by a user, associated with one or more successful authentication events of that user.

Care Provider Stories

Use a Native Application with the Electronic Health Record

As a care provider, I want the ability to enable a native application to utilize the electronic health record on my behalf, so that I may take advantage of an entire marketplace of applications that help provide better care to my population.

Example: The user downloads an application that helps visualize a specific patient's risk of heart disease based on their health record, and uses it in the office to help convey the various ways the patient can lower their risk.

Use a Web Application with the Electronic Health Record

As a care provider, I want the ability to enable a web application to utilize the electronic health record on my behalf, so that I may take advantage of web applications that help provide better care to my population.

Example: A doctor visits a web application provided by a drug company that helps to determine if a particular new drug is appropriate for treatment, based on a number of factors. The doctor chooses a patient to have evaluated; the web application retrieves information about the patient and presents its analysis to the doctor.

Instructions When Attempting to Use Unregistered Applications

As a care provider, I want clear instructions to be presented to me when I attempt to use a client application that has not been authorized by my organization, so that I may request it be made available by my administrator.

Example: A specialist learns about a new tool that can help diagnose a particular condition. The specialist downloads and runs the application, but finds it isn't registered with their organization. The system alerts the specialist as to the next steps to request it to be registered.

Share My Identity with Client Applications

As a care provider, I want client applications to know my identity as a provider so that such applications can give me a personalized experience. Such an experience might include the ability to customize behaviors for future visits, the ability to generate content personalized with my name, the ability to receive text messages when results are available, etc.

Example: As a doctor, I utilize a web application that makes recommendations to follow-up on high-risk patients. This site remembers which individuals I've prioritized to follow-up on from my previous interactions, regardless of which device I access it from.

Care Consumer Stories

Download and use a Native Application with the Electronic Health Record

As a care consumer, I want the ability to download and install a native application and allow it to access my health record so that I may benefit from a wider selection of tools for managing my health.

Example: A doctor prescribes the use of a blood sugar monitoring tool that can integrate with my mobile device, which in turn can update my health record automatically with routine measurements.

Use a Web Application with the Electronic Health Record

As a care consumer, I want the ability to grant a web application access to my health record so that I may benefit from a wider selection of tools for managing my health.

Example: A dietician recommends the use of a specific web application that provides dietary recommendations based on current medications.

Use a Shared Device with Other Consumers

As a care consumer, I want to be able to securely access my health record from any device I personally trust so that I have opportunities to use my electronic health record even if I do not have a mobile device.

Example: An outpatient clinic provides a kiosk for patients to log in to update their medical history and current medications. The user logs in, accesses their information, and then logs out. Applications the patient accessed should honor/acknowledge the user has logged out and not serve the previously logged-in patient’s resources to other users.

Consent to Risks

As a care consumer, I want any sharing of my personal health information or personally-identifiable information that may be considered risky to require my approval and be presented in plain language so that I may better secure my identity and privacy.

Example: An application suddenly and unexpectedly asks to access a consumer’s entire medical record. Having not invoked any actions that would have precipitated such a request, the consumer chooses to deny it.

Instructions When Attempting to use Applications Unknown to the Health Record

As a care consumer, I want clear instructions when I attempt to use a client application that is unknown to the health record system so that I may request that it be added for use.

Example: An individual attempts to use a health visualization tool, but it is not registered with their care provider’s system. A page appears with a link to a help desk tool that allows the individual to request it be approved for use. The care provider then reaches out to the developer of the client application to get the information necessary to securely register it with the electronic health record.

Identify Myself to Client Applications

As a care consumer, I want my identity to be known to selected client applications so that my choices, application settings, and any work I have done can be made available to me on other devices.

Example: The user logs in to a web application that provides feedback on the progress of their latest lab results. The site tracks which lab results the user has previously seen, and thus needs a way of uniquely identifying users from visit-to-visit.

Use Client Applications on Behalf of Another

As the legal guardian or custodian of another individual, I want the ability to use client applications on behalf of another individual so that I do not have to track multiple usernames and passwords for those in my care.

Example: A parent signs into the health record site, then switches context to their daughter, who is a minor. The parent then launches a diabetes tracking application, utilizing the daughter's identity. Any subsequent client application or web application visits are also performed as the daughter's identity, until the parent logs out (either explicitly or by switching context back to themselves or another child).

Terminate Previous Consent

As a care consumer, I want to be able to terminate authorization for a client that I previously consented to, so that I may better protect my privacy. This ability should be provided by the electronic health record.

Example: An individual allows a free fitness tracking application that monitors the individual's weight. After receiving numerous unsolicited, automated messages offering fitness and diet tips the user decides to terminate the application's ability to monitor their weight.

EHR Administrator Stories

Register a Client Application

As an administrator, I want the ability to register client applications for use with the electronic health record to ensure that data is sent only to applications that my organization has approved.

Utilize Registration Data from a Trusted Party

As an administrator, I want the ability to configure my system to use applications that have been pre-registered by another trusted party (such as the EHR provider) so that my organization can automatically make use of client applications that have been certified for use.

Configure Messages for Users for Unregistered Applications

As an administrator, I want the ability to configure messages for users when they attempt to use an application that has not yet been registered so that I may direct them to the appropriate contact or venue for requesting the application be certified and added.

Configure Messages for Users for Blacklisted Applications

As an administrator, I want the ability to configure messages for users when they attempt to use an application that has been explicitly revoked from using services so that I may alert them as to why.

Deploy a Monitor that Utilizes a Client Application

As an administrator, I want the ability to set up specific devices to access the electronic health record via a client application for purposes of providing an always-on display of information that are secured by physical location.

Example: The nursing staff for the emergency room needs an always-on display of the status of registered patients. The administrator provisions a device configured to load the appropriate client application. Instead of a user authenticating the device at start-up, instead the administrator uses tools provided by the EHR provider to configure the device to authenticate itself using some form of "service user".

Register "B2B" Client Applications

As an EHR administrator, I want the ability to register client applications that authenticate/act on the behalf of another business entity so that I may leverage value-added services provided by another organization.

Example: A third party provides a service that continually monitors reported symptoms in visits across clinics, aggregates the information, and sends an automated alert when a possible outbreak of a condition such as measles is detected. The administrator configures its endpoints, credentials, and individuals that are authorized technical contacts for the third party.

Risk and Compliance Officer Stories

Restrict Authorization Limits of Client Applications

As a risk and compliance officer, I want the ability to set restrictions on how client applications are able to access EHR resources so that I may ensure the risk is tolerable for my organization.

Example: The compliance officer configures restrictions on a web application, preventing it from accessing patient records on behalf of a care provider once the care provider has logged out.

Restrict Usage of Client Applications to Authorized Users

As a risk and compliance officer, I want the ability to restrict which users are capable of accessing specific client applications, so that I may ensure such users have had appropriate training or are equipped with compliant devices.

Suspend Access of a Client Application

As a risk and compliance officer, I want the ability to suspend a client application's access so that I may stop any data exchanges with a client application whose code or services have been compromised.

Suspend Authorizations Granted for a User Session

As a risk and compliance officer, I want the ability to suspend any authorizations granted during a user's session, so that I may respond to security incidents that involve the compromise of a user's device or credentials used for authentication.

Restrict Unattended Access

As a risk and compliance officer, I want the ability to restrict which client applications and authorizations that can be acquired that allow access to data on behalf outside of the user's session, so that I may limit the organization's risk exposure for a compromised application or device.

Require Explicit User Approval for Authorization Categorized as "High Risk"

As a risk and compliance officer, I want the ability to require care providers to explicitly approve certain authorizations that are requested by client applications so that I may prevent accidental usage of EHR services that expose the organization to higher risks.

Example: A physician uses a personal device to access a patient's record in the EHR. Rather than blocking access, the authorization server asks the physician to vouch that the device conforms to the organization's workstation security policies, and to confirm that the application was installed via links provided by the organization. The risk and compliance officer enables this feature for a select few physicians whom have had in-person training.

Limit Authorization of Certain EHR Services to Particular Devices

As a risk and compliance officer, I want the ability to set restrictions on certain services such that they may only be invoked from specific known devices so that I may restrict the location in which certain sensitive workflows occur.

Example: An organization restricts access to web services that can access an individual's financial information to specific workstations used by registration staff in the hospital. The EHR provides tooling that allows administrators to register such devices such that they are authenticated during the authorization process.

EHR Developer Stories

Single Mechanism for Service Authorization

As an EHR developer whom is developing a resource server, I want a singular mechanism to verify that an authorization server has granted authorization to a given client so that I may reduce the complexity of the implementation of the server.

Support for Authentication Provided by Native Client Applications

As a developer of an authorization server, I want client applications to obtain authorization in such way that does not prohibit my ability to use native components during the authentication process on the workstation, so that I have flexibility in using components that provide security.

Example: An authorization server invokes a native component that communicates to a hardware component, returning an identifier of a location sensor attached to the mobile device. The authorization server continues to track the location of the mobile device in the facility. If the device leaves the facility, it terminates any authorizations for client applications that were obtained with it.

Client Application Developer Stories

Simplified Interface for Requesting Authorization

As a developer of a client application, I want a simplified contract with the authorization server that shields me from the complexity of user authentication, session management, authorization logic, device identification, and any other private implementation details that would require proprietary integration with disparate EHR systems.

Ability for Client Applications to Access Services on Behalf of a Business Entity

As a client application developer of a client application, I want the ability to have access to some services by authenticating my application directly, so that I can develop applications that can augment the electronic health record without user interaction.

Example: A third party provides a service that continually monitors reported symptoms in visits across clinics, aggregates the information, and sends an automated alert when a possible outbreak of a condition such as measles is detected.

Client Application Developed using JavaScript Only

As a client application developer of a client application, I want the ability to obtain access on behalf of a user using JavaScript without server-side componentry so that I can provide embeddable web applications that are easily hosted.

Risks and Threats

Enabling the use cases described in the previous section requires an analysis of the types of risks and threats that would be present by enabling some or all of the cases. This section describes possible risks and threats that should be addressed in such an ecosystem.

Compromised User Authentication or Sessions

This section notes various risks and threats to user authentication and authorization.

User is "Phished"

A common threat to online systems that targets users are "phishing[2]" attacks. In such an attack, a malicious party attempts to gain access to the user's credentials to access an online system. In most environments, users have been conditioned to automatically respond to authentication prompts invoked by applications (as opposed to authentication or unlock mechanisms provided by the operating system). These environments present an issue, as a malicious application need only to present an authentication prompt that appears similar to the one a user would normally expect. This is easier to exploit on mobile platforms due to the lack of visual indicators presented to a user as to which application and/or website they are entering data into.

Even if caught and addressed, the malicious party could utilize access obtained with the user's credentials until such access expires or is explicitly revoked. Additionally, such a party could continue to use a session established to the EHR or services unless mechanisms exist to terminate such sessions. A user may be completely unaware of the fact their credentials were stolen in this fashion.

Malware on the User's Device

Another avenue to illicitly gain access to a system is via the installation of malicious software on a user's device. Over the years, such attacks have come through a number of avenues. These include malicious hardware such as USB devices, browser-based attacks that exploit software vulnerabilities, and social engineering tricks to get users to install the malicious software themselves. Once installed on a device the malware could obtain the user's password, steal the user's session or authorization tokens, or masquerade as one of the user's authorized applications. Such activities may go unnoticed by the user.

Device or Workstation is Unsecured

A common concern in healthcare institutions is handling scenarios where PHI is left exposed when a user leaves a workstation or mobile device unattended. This allows the opportunity for information to be accessed by an unauthorized party.

[2] http://www.fraud.org/scams/internet-fraud/phishing

Application Credentials

Not all applications will need to authenticate themselves to the EHR, but for those that do, a safe mechanism must exist for them to authenticate. This section discusses the risks and threats associated with such credentials.

The most common means of authenticating another party are via the use of a "symmetric" credential or an "asymmetric" credential. A symmetric credential is one whose secret is known by both parties, such as a traditional key (such as a username) and secret (i.e. a password). An asymmetric credential is one in which two pairs of keys exists – a private key and a public key. Cryptography is used by the holder of the private key as proof of possession; holders of the public key may perform a reverse operation to verify the authenticity of such proof.

Accidental Disclosure of Symmetric Credentials at Initial Exchange

After an application or service that needs the capability to authenticate is initially vetted, it will need credentials to authenticate. If these new credentials are exchanged via insecure methods (such as unencrypted e-mail, or stored in an e-mail account that is later compromised), a malicious party might be able to obtain them.

Accidental Disclosure of Symmetric Credentials While Authenticating

When an application sends its credentials to the EHR, safeguards must exist to ensure that it is doing so via a trusted, verified channel. Some examples:

- If the credential is not sent with some mechanism to prevent eavesdropping (i.e. encryption), an observer with physical access to the network could capture the secret.
- If using an encryption mechanism that is negotiated when contacting the EHR, the application must verify the identity of the EHR. Otherwise, an actor with control over the network could insert himself (a "man-in-the-middle" attack[3]) and intercept the secret.
- If a fundamental misconfiguration occurs on the part of the administrator, the credentials could be sent to the wrong party. This could be the result of incorrect configuration data in the application, or by blindly accepting an HTTP redirect that results in the credentials being sent to a foreign server.

Theft by Insider

Often times, attacks come from insiders whom have access to sensitive materials. Credentials are generally available to the administrators who initially configure them and to server administrators that have root access to devices that utilize the credentials.

Social Engineering Attacks

A malicious party may attempt to impersonate the administrators or developer of a client application. A successful subversion could allow such individuals to gain access to any PHI that the client application would normally have access to.

[3] https://www.owasp.org/index.php/Man-in-the-middle_attack

Authorization Tokens or Session Secrets

Many platforms allow clients to exchange authentication for a temporary secret or token that allows the client to access services without the need to respond to an authentication challenge on every request. This section discusses the risks associated with the usage of such tokens and secrets.

Disclosure to a "Counterfeit" Protected Resource

A client application could be tricked into sending valid secrets to a malicious party that is masquerading as a protected resource that the client wishes to access. Such an event could result from a misconfiguration (e.g. a typo when manually registering the location of an EHR's services), or from protocol-level vulnerabilities that allow external parties to direct a user's system to attempt authentication with any public URL.

Accidental Disclosure while Accessing a Resource Server

Similar to authentication credentials, a client must ensure they are communicating with a trusted resource server over a secure channel, and not accidentally send tokens or secrets to untrusted parties as a result of misconfiguration of the resource server.

Theft by Insider

Similar to application credentials, a client/server application could have authorization tokens or session secrets stolen by an insider with access to the server.

Risks Associated With the Use of TLS for Transport Encryption

This section discusses common risks and threats known when using Transport Layer Security, or "TLS"[4]. TLS is the security mechanism for HTTPS (Hypertext Transfer Protocol Secure)[5], which is the industry-standard way for protect data being communicated via HTTP. As the majority of web-based security protocols are built on the security that TLS provides, it must be included in this risk analysis. TLS provides two important features:

- Ensures confidentiality of the data, by authenticating the remote party's identity via X.509 Public Key Infrastructure (PKI)[6]. Once authenticated, a cipher is negotiated to encrypt the transmission of data between the two parties, ensuring confidentiality.
- Ensures integrity of the data, by cryptographically signing transmitted data. Any attempt to tamper with the data by manipulation of the cipher-encoded data is detected by the TLS protocol.

Compromised Certificate Authorities and/or Authority Non-Compliance

To facilitate the authentication of remote services, PKI utilizes a "trust store", which contains a list of the certificates of trusted certificate authorities. Such trust stores are used by end user devices to enable https web browsing, and to validate the authenticity of software. The authorities included in this are

[4] TLS 1.2 http://tools.ietf.org/html/rfc5246
[5] HTTP over TLS http://tools.ietf.org/html/rfc2818
[6] Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework http://www.ietf.org/rfc/rfc3647.txt

generally capable of vouching for the identity of any system, even if no business relationship exists between the two parties. The contents of trust stores are generally vetted by the operating system or browser manufacturer[7,8] to ensure such authorities act with good faith and have taken the necessary precautions to secure their services from threats. Even with such rigor in place, security incidents have occurred in the past. Such events include authorities being compromised to issue fraudulent certificates[9,10] and authorities issuing certificates in bad faith[11].

In many circumstances, EHR users (either patients or providers) will have access to EHR services over untrusted networks. An attacker with access to fraudulent certificates could compromise the communication chain, leading to any of the following repercussions:

- Compromise of the user's credentials.
- Compromise of the application/service's credentials.
- Compromise of authorization tokens or sessions.
- Ability to read PHI and interact with EHR services on behalf of the user.

While browser and operating system manufacturers generally react to known compromise events, older devices may not be receiving such updates[12], making them prime targets for such attacks.

Compromised Private Keys

If an application/service's private key for their web application certificate has been compromised, devices are left vulnerable as described previously. Certificate authorities offer revocation capabilities in the form of revocation lists[13] and web services[14], but some applications may not correctly make use of these features. Furthermore, if a certificate trusted by an application or service is self-issued, such revocation functions may not exist at all, thus requiring removal from the application's trust store.

Protocol Vulnerabilities

With any security protocol, vulnerabilities may be discovered that require action by the service provider, client application, or both. For TLS, a number of problems have been identified in recent years[15]. Effectively, this creates a "moving target" for security in frameworks. What may be secure and interoperable one day may be found to be completely insecure the next. TLS, in particular, is

[7] Mozilla.org - Maintaining Confidence in Root Certificates
https://wiki.mozilla.org/CA:MaintenanceAndEnforcement
[8] Microsoft Root Certificate Program - http://msdn.microsoft.com/en-us/library/cc751157.aspx
[9] http://www.computerworld.com/article/2510951/cybercrime-hacking/hackers-spied-on-300-000-iranians-using-fake-google-certificate.html
[10] http://blog.mozilla.org/security/2011/03/25/comodo-certificate-issue-follow-up/
[11] http://www.computerworld.com/article/2486644/security0/other-browser-makers-follow-google-s-lead--revoke-rogue-certificates.html
[12] http://theunderstatement.com/post/11982112928/android-orphans-visualizing-a-sad-history-of
[13] Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile
http://tools.ietf.org/html/rfc5280
[14] X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP
http://tools.ietf.org/html/rfc6960
[15] http://www.techsupportforum.com/20098-ssl-and-tls-vulnerabilities-are-we-safe/

susceptible to this – currently three minor versions exist, with a fourth currently in development. Furthermore, a myriad of “cipher suites” are possible for securing communication offering a range of different cryptographic techniques for encrypting the data and securely exchanging protocol secrets.

To complicate matters, a balance must be struck between security minimums and what can generally be supported in the market. For example, as of this writing, approximately 8% of the install base of Android devices is currently on version 2.x[16], which does not support TLS version 1.2.

Cloned Native Application Trojans

For web applications, a three-party trust can be established via the use of https and by requiring those TLS-authenticated web applications to authenticate when obtaining authorization. This prevents other unaffiliated websites from “cloning” the appearance of a valid website and obtaining the user’s authorization fraudulently. For native applications, however, no similar trust model exists. When an application is downloaded and installed from an application store or installed directly, it does not possess any method of proving its own identity.

Such “cloned” applications provide a possible avenue of obtaining PHI on behalf of the authenticated user without their knowledge. Where the given user device is not tightly controlled by an organization, this may place the responsibility on the user to protect the operating environment from such malicious software. This may, in turn, place training requirements on the EHR user; for consumers, it may be a click-through warning educating them to the risks.

Storing Sensitive Data on the Device

Health information, session secrets, and authorization tokens are all sensitive pieces of information that must be kept private and are properly sandboxed from other applications.

Device Sharing

Today, a majority of American households have connected devices[17]. Often times, these devices are shared between family members. In care organizations, a pool of devices may exist to support shift workers. For users that are properly trained to log off of an application, the risk still exists that other applications running may retain data or access unless designed to have “awareness” of the user session, or unless the user explicitly ensures that such applications are inaccessible to the next user (some operating systems have support for multiple users). The HIPAA technical safeguards[18] list automatic logoff as an addressable requirement for this reason.

Furthermore, if it is necessary to store PHI on the device (for example, a mobility application for hospice care) such data should be encrypted in a way that prevents off-line cracking.

[16] http://developer.android.com/about/dashboards/index.html#2015
[17] http://recode.net/2014/11/18/more-than-90-percent-of-u-s-households-have-three-or-more-devices-pinging-the-internet/
[18] HIPAA Security Series Technical Safeguards 164.312(a)(2)(iii) - http://www.hhs.gov/ocr/privacy/hipaa/administrative/securityrule/techsafeguards.pdf

Finally, shared devices pose a risk to web-based authentication systems. Web browsers increasingly offer to remember passwords for web applications, regardless of the wishes of the web application owner. As the boundaries continue to be pushed on password-saving features, a greater responsibility is put on users within provider organizations to correctly protect their device, or on the enterprise to lock down the features of devices allowed to access EHR services. For individuals accessing their own health records, education around the use of browser features such as in-private browsing may be prudent as well.

Exposure of Device Content to Unauthorized Parties

Many mobile devices present themselves as USB storage when unlocked and connected to a PC via USB. Most will prompt the user when this occurs, asking how the connection should be used (charging only, access to pictures and music, full access, etc.). Care should be taken to utilize secure storage mechanisms for any encryption keys that are inaccessible via these means.

Storage of Session or Authorization Tokens

Similar to encryption keys, session or authorization tokens should be suitably secured. Native applications should leverage features provided by the platform to secure passwords, session identifiers, or authorization tokens that prevent accidental disclosure via error reporting mechanisms, such as application dumps. Such mechanisms also protect against writing such secrets to disk during memory swap operations, which could reveal secrets to an attacker who obtains a lost or stolen device.

User Impersonation by Applications to Other Web Applications

Where a web application utilizes an EHR for determining the identity of the user, the possibility exists that additional information is being provided by that site that is outside of the data it is currently authorized to access. Examples of such information may include data owned by the user, information previously accessed on behalf of the user by the application, or PHI derived from other sources. Depending on the design of the user authorization and identification process, this may allow an opportunity for a second web application that participates with the EHR to reuse an authorization it has obtained on behalf of a user to impersonate the user and gain access to the additional information available at the first web application. This is also known as the confused deputy problem[19]

Privacy and Anonymity

Individuals utilizing applications that leverage their personal health information may wish, where possible, to do so anonymously. This may require that applications use identifiers for users that are unique per application, as to prevent correlation between applications that might result in identification of the user.

Denial of Service / Excessive Usage

Poorly written or malicious applications can have detrimental effects on services provided by an EHR. When an EHR's services are made available outside of an organization's private network (such as to enable individuals to access their own health records) the possibility for attack is opened as well.

[19] http://stackoverflow.com/questions/17241771/how-and-why-is-google-oauth-token-validation-performed

Excessive usage, whether intentional or accidental, can severely impair a system. The authorization implementation should be aware of key facts that allow EHR services to adequately protect themselves.

Framework Recommendations

To satisfy the described user stories, a framework is needed that allows a client application to prove that it is acting on behalf of an individual, and optionally that the individual explicitly approved such actions. Furthermore, the framework must be capable of restricting which client applications are allowed to act on behalf of users, and impart restrictions on what actions it can perform on behalf of an individual. It must also provide a standard way for client applications to obtain access on behalf of a business entity (for “business-to-business” service calls). Finally, the framework must be amenable to web applications, native applications, and RESTful services.

The OAuth 2 framework, as of this writing, is the only framework that is designed to handle all of these use cases for authorization between a user, service, and application via both browser and native applications[20], as well defines profiles suitable for use with RESTful web services[21]. While similar in features, the SAML 2.0 specification from OASIS[22] does not currently define a mechanism for authorization decisions (XACML-based or otherwise) to be utilized with REST services, nor are mechanisms described for which it could interoperate with native clients.

OAuth 2 works well on a myriad of platforms because it relies on URI-based redirects to enable communication between applications. Such redirect mechanisms work well for both switching between different web applications within a browser, as well as switching to native applications installed on the user’s device.

Mapping OAuth 2.0 Grants to Use Cases

OAuth 2 defines four means of obtaining authorizations, or “grants”. Each is intended to enable specific use cases, two of which map to use cases described in these recommendations. In addition, the protocol’s grant mechanisms are extensible, allowing it to service other use cases.

Commonalities of Grant Types

In each of the grants, a **client** attempts to obtain an **access token** from an **authorization server** either directly or via an individual’s **user-agent**. The access token enables the client to make calls to REST services on the **resource server** that require authorization. These terms are defined by the RFC as follows:

- **Resource Server** - The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens.
- **Client** - An application making protected resource requests on behalf of the resource owner and with its authorization. The term "client" does not imply any particular implementation

[20] The OAuth 2.0 Authorization Framework - http://tools.ietf.org/html/rfc6749
[21] The OAuth 2.0 Authorization Framework: Bearer Token Usage - http://tools.ietf.org/html/rfc6750
[22] Security Assertion Markup Language Version 2.0 - https://www.oasis-open.org/standards#samlv2.0

characteristics (i.e., whether the application executes on a server, a desktop, or other devices).

- **Authorization Server** - The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization.
- **Access Token** - Access tokens are credentials used to access protected resources. An access token is a string representing an authorization issued to the client. The string is usually opaque to the client. Tokens represent specific scopes and durations of access that were approved by the resource owner, and are enforced by resource servers.

While not directly defined by the RFC or by any official standards body, a user-agent is generally accepted to mean either a web browser or the client component of a client-server application.

In each workflow, the client passes a set of requested **scopes**. A scope represents a permission to act upon a resource in some fashion. This provides the authorization server the opportunity to determine if the client is allowed to make such requests, and in some cases, if the user is willing to approve the requested scopes. An authorization server may decide to fulfill only part of the requested scopes (perhaps based on user feedback), none, or all of the scopes. Once approval is received, further steps may be required, depending on the type of grant mechanism being utilized. At the end of the grant processes, the client will either receive an error indicating why the request was denied (all scopes were rejected), or an authorization token that represents the authorization for one or more of the requested scopes.

Authorization Code Grant

This grant enables a three-party workflow between the user-agent, an authorization server, and a client. This enables use cases where one or more of the following is required:

- The authorization server must identify the authenticated user to make an authorization decision.
- The user must provide approval for the client to obtain authorization to access the resource server.
- The kiosk needs to be identified by the authorization server to make an authorization decision.
- The authorization server wants to frequently verify that the client is still a participant in the usage of a given token and utilizes a refresh token to do so.

A distinguishing feature of this grant is that the client can be required to authenticate to complete the workflow. Because native applications generally cannot protect a global client secret, this authentication only provides protection to web applications or applications with server components that are capable of protecting the client secret. By requiring authentication to obtain the authorization token passive observers of the user’s browser history (third party browser integrations, other applications on the device) are prevented from obtaining the authorization token.

The following diagram from the OAuth 2 RFC[23] illustrates this workflow:

```
+----------+
| Resource |
|   Owner  |
|          |
+----------+
     ^
     |
    (B)
+----|-----+          Client Identifier      +---------------+
|         -+----(A)-- & Redirection URI ---->|               |
|  User-   |                                 | Authorization |
|  Agent  -+----(B)-- User authenticates --->|     Server    |
|          |                                 |               |
|         -+----(C)-- Authorization Code ---<|               |
+-|----|---+                                 +---------------+
  |    |                                         ^      v
 (A)  (C)                                        |      |
  |    |                                         |      |
  ^    v                                         |      |
+---------+                                      |      |
|         |>---(D)-- Authorization Code ---------'      |
|  Client |          & Redirection URI                  |
|         |                                             |
|         |<---(E)----- Access Token -------------------'
+---------+       (w/ Optional Refresh Token)
```

With this workflow, the client application utilizes a user-agent (the browser) to provide the resource owner (the individual user) the ability to interact with the authorization server. During this interaction, the user must authenticate to the authorization server (or have previously been authenticated and established a session) and optionally approve the client's request to act on their behalf. The result of the transaction is a token that represents the client's "authorization" to call protected resources on the user's behalf.

This workflow exempts the client from handling the user's credentials directly – this improves the overall security posture by ensuring that limits by may be imparted on client applications to what actions they may take on behalf of the user. Furthermore, it gives the authorization server flexibility in the types of authentication that can be used (innovation can occur without the client application needing to support new forms of authentication).

[23] http://tools.ietf.org/html/rfc6749#section-4.1

Client Credentials Grant

This grant is similar to the authorization code grant, except that the user-agent is not present in the workflow. Instead, the client uses its own credentials directly to obtain an authorization token. This enables workflows where a third-party service is provided access to services directly for processing outside of user-driven workflows.

Extension Grants

OAuth 2.0 also provides for extensibility; custom extension grants may be defined by implementations. Such extensions allow the EHR to define its own mechanism for obtaining authorization tokens on behalf of itself. This allows it to use the same authorization mechanism for REST services that other third-party clients utilize, minimizing the number of authorization models needed to be supported by the REST services. For example, a web-based EHR might be able to utilize the user's session cookies to obtain an authorization token, allowing the EHR to then make calls to the REST services via XMLHttpRequest in the browser.

Mapping OAuth 2.0 Scopes to Use Cases

Scopes provide the capability for a client to request specific access rights on behalf of the user (or itself). This provides three useful features:

- The specific authorization needs of the client can be described to the authorization server.
- The authorization server can, in turn, describe to the user what is being requested where explicit approval is required.
- Authorization tokens can be limited to a subset of a given user's privileges.
- Applications can request fewer scopes than they are entitled for and delegate usage of authorization tokens to the client or to another sub-component of their system.

Scope Fulfillment

Per the OAuth2 specification, an authorization server may choose to only fulfill a portion of the grants requested by a client. Furthermore, a client may omit its list of grants and defer to a pre-configured behavior of the authorization server to choose the grants it will receive. If no grants are authorized, then it is a requirement of the authorization server to return an error to the requesting client indicating that the client may take no actions on behalf of the user. The following is list of possible outcomes:

- A client requests no scopes; the authorization server grants pre-configured scope(s) allowed for the client.
- A client requests no scopes; the authorization server returns an error indicating no scopes were authorized.
- A client requests scopes and is provided all scopes requested.
- A client requests scopes and is provided a subset of scopes requested.
- A client requests scopes and is provided additional scopes beyond those requested and zero or more of the requested scopes.
- A client requests scopes; the authorization server returns an error indicating no scopes were authorized.

Different use cases may dictate whether the authorization server prompts the user for explicit approval, and if it allows for selective choice of approving individual scopes. OAuth client applications should be designed to inform the user if it does not receive scopes that are necessary for the application to function. In this scenario, the client may not be able to instruct the user appropriately how to resolve the situation. An authorization server that receives a request with the same scopes within a short timeframe for a single user's session should inform the user of any potential problems (such as information on how to contact the local administrator to pre-approve additional access for the application). This method provides both a break-point for infinite loops in the authorization request process, and a reliable method to direct the user towards assistance in resolving the issue.

EHR administrative tooling will need to provide capabilities of registering applications, choosing pre-authorized scopes, and designating which scopes and/or applications require explicit user approval based on type of user (e.g. care provider versus patient, etc).

Scope Values

There are a myriad of possible scope values that might be recognized by an EHR, but no central registry exists that defines them. As part of offering open services on top of the EHR, some form of governance model will need to exist for defining the possible scope values. The following should be considered when designing the values:

- A concrete definition of the scope that clearly defines the boundaries of what it authorizes a client to access.
- Versioning mechanisms to allow for updates to the semantics of a scope over time.
- The governance model for adding / modifying / deprecating scope values.
- A name-spacing mechanism to prevent collisions with other specifications that utilize scopes.

OAuth 2.0 Bearer Token Lifecycle

Client applications may have differing requirements with respect to how long they need to act on behalf of the user. This section discusses each of the possible token lifecycles and associated security precautions needed for each.

Short-Lived Access Tokens

A short-lived access token can be defined as a token whose duration is equal or less than the maximum duration of any inactivity-based limit in an EHR. No revocation mechanism is required for a short-lived token; it can be assumed that the lifecycle is short enough that if acquired by an unauthorized party that it would expire nearly as a quickly as a user or administrator could respond to the threat.

A resource server receiving such a short-lived token would still need to verify the scopes claimed in the token. Additionally, the authorization server may only be verifying the authorization of a given client to act on behalf of a user – in these scenarios, a resource server will still need to ensure that the authenticated user is authorized to perform a given action.

Short-Lived Grants

In many cases, it may be valuable to issue access tokens for short durations (measured in minutes), but allow for the client to obtain a new short-lived token with the original grant while the user's session is still valid. Reasons for a session becoming invalid include:

- The user logs off and makes the device available to other users.
- A user logs in and chooses to terminate sessions from other devices (having not actively chosen to log off from a previous device).
- An administrator terminates a user's sessions due to a security incident (e.g. the user's device or account information was stolen).
- The user's session is automatically terminated due to inactivity.

A **refresh token**, as described by section 6 of the OAuth 2.0 specification[24], should be used for such access. This forces the client application to obtain new access tokens on behalf of the user on a regular basis and allows the authorization server to discontinue providing authorization for a terminated session. This also allows a client application to determine if a user's session is still valid. In the event of the loss of a device, or suspected compromise, avenues should be provided to both the end user and to an administrator to revoke previous, active sessions.

Infinitely-Lived Grants / Offline Access

Some clients need the ability to act for an unbounded amount of time on behalf of a user, often referred to as "offline access". Clients utilizing such tokens should also utilize refresh tokens. This provides several benefits.

- For clients that are storing many offline access tokens issued for different users, the authorization server can force the client to authenticate when refreshing the token. This prevents access token that may have been accidentally disclosed from needing revocation and re-issuance; a client that is completely compromised need only change its authentication credentials. Without such a refresh mechanism, it would require all of the access tokens to be re-issued, possibly affecting hundreds or thousands of users.
- Even for clients without credentials (i.e. "public clients") that only store a single user's access token, the refresh token mechanism can provide some value between the authorization server and its associated resource servers. Resource servers might be able to rely on some or all of the authorization checks being performed by the authorization server due to the short-lived nature of each access token, leading to greater operational efficiencies.

In the event a user's access was compromised, tools must be provided to the end user and administrators to review and revoke such long-lived access tokens that may have been issued during the period of compromise (such tokens were issued without the user's approval). As such, the tooling should provide the dates and times at which the token(s) were issued. Ideally, if long-lived access tokens are issued that will be stored on a device possessed by the user, a recognizable identifier for the

[24] The OAuth 2.0 Authorization Framework - http://tools.ietf.org/html/rfc6749

name of the device would be captured during the authorization process. This would allow for a user or administrator to revoke any and all tokens issued to a given the device if loss or theft occurs.

Requesting Offline Access

Additionally, a method is needed for the client to convey its requirement to obtain authorization to act in an offline capacity for the user. When requesting that a grant be infinitely lived, a client should use a special scope named "offline_access" such that the authorization server can identify and respond to such requests.

Registering Applications

For authorization code grants, a methodology is needed to ensure that such grants are only disclosed to known, approved parties.

Redirection URI

An administrator must register one or more redirection URIs associated with a given client application. This ensures that other websites accessed by a user cannot arbitrarily request access. This protection is enforced by the user agent – it is assumed if the user's device or browser has been compromised, an attacker would already have sufficient means to gain access on behalf of the user.

Client Identifier

In addition to the redirection URI, it's necessary for the client application and authorization server to agree on a client identifier in which the client will use to identify itself during the request. Unlike some OAuth implementations, it is desirable to allow both public and confidential clients to publish their own globally-unique client identifiers for their application. In doing so, it allows an ecosystem where an EHR administrator could register any publically-available client application for use with their EHR without intervention on the part of the owner of the client application.

Identity Assurance

When an EHR administrator registers a new client application, there will be a need to verify that the app provider is an authentic business entity. The use of public key infrastructure (PKI) is one way to help assure the identity of an external party. Instead of performing manual identity assurance of an individual requesting client set-up or configuration changes, administrative tooling can leverage information present in Extended Validation certificates[25] that are used to protect the given redirection URI. This process provides an EHR administrator proof of identity of the client application provider, which includes the verified corporation file number of said party. Alternatively, an organization may perform its own business identity verification mechanisms.

[25] CA/Browser Forum Guidelines For The Issuance And Management Of Extended Validation Certificates - https://cabforum.org/wp-content/uploads/EV-V1_5_2Libre.pdf

Client Authentication

For some authorization code grants and all client credentials grants, a client requires a method of authenticating to the authorization server is required. The following method is proposed as a mechanism to protect against insider theft and accidental disclosure of secrets, as well as to incorporate practices that can be used to minimize social engineering risks. To achieve these goals, a combination of public certificates and signed JSON Web Tokens[26,27] are prescribed below, with rationale provided.

Registration

Under this model, the administrator of a client application or web application provides a JSON Web Key Set URL where one or more JSON-encoded public keys may be retrieved from for use in verifying signed JSON web tokens during the registration process. Similar to the normal registration process, the registration tooling may convey identity information derived from the certificate used in fetching the https:// URL.

Authentication

When accessing the token endpoint, the client application uses a JWT bearer token, signed with one of the keys advertised in its JSON Web Key Set. Such authentication should be required for any client credentials grant. It is recommended that authentication also be required for any client that stores refresh tokens

Credential Revocation and Rotation

When using public keys with signed JSON Web Tokens, the public key set's location shall be conveyed via the "jku" parameter. The authorization server compares the location of this parameter against the registered JSON Web Key Set URL for the client. If the values do not match, the token should be rejected. A client may, at any time, revoke one or more of its public keys, thus preventing an authorization server from verifying any tokens issued with that key. A client may advertise new public keys, providing a mechanism for rotating public/private key-pairs over time without downtimes.

This mechanism allows a client to quickly re-issue / revoke credentials with all authorization servers without having to individually contact each. No out-of-band exchange of symmetric keying material is required, thus reducing the risk of the private credentials being compromised. Furthermore, the established PKI model protects the JSON Web Key Set location using the longer-lived x509 public certificate, which also has a revocation model in the event of compromise.

Authorization servers should take all precautions required by the JSON Web Signature specification for verifying the authenticity of the JSON Web Key Set endpoint.

[26] JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants (Draft 12) - http://tools.ietf.org/html/draft-ietf-oauth-jwt-bearer-12
[27] JSON Web Signature (JWS) (Draft 41) - https://tools.ietf.org/html/draft-ietf-jose-json-web-signature-41

Interoperability Requirements

Per section 5 of the JWT bearer token specification[28], the following requirements are to be observed by clients and authorization servers:

- The issuer and audience identifiers will be the same for tokens signed for client authentication. These identifiers will be issued by the authorization server at time of client registration.
- Digital signatures will be verified using public keys, advertised at a JSON Web Key URL. This URL will be provided at client registration, and will be conveyed in the token.
- Authorization servers, at a minimum, must support both the following JSON Web Algorithms[29]. Clients must, at a minimum, support one.
    - RS256
    - ES256
- Clients must provide an Expiration (exp) claim that is no greater than five minutes from time of issuance.
- Authorization servers must allow for 3 minutes of clock skew when validating expiration (exp) claims.
- Clients must use the token endpoint as the value for Audience (aud). Authorization servers must verify its token endpoint is in the audience claim.

Additionally, clients must also perform the following:

- Provide the claims Issuer (iss) and Suject (sub), with its client_id as the value.
- Provide a unique value for the JSON Web Token ID (jti).

OpenID Connect and Use Cases

OpenID Connect (OIDC) describes itself as *"…a simple identity layer on top of the OAuth 2.0 protocol"*[30]. In its simplest form, it provides an additional token known as an **id_token** along with an access token as part of the authorization grant process. This token contains information conveying the identity of the individual, as well as information to validate that the token is not being misused in some fashion (e.g. replaying an authorization grant to a separate system to impersonate a user).

The proper use of this protocol, however, has practical and functional implications, most of which affect administrators and implementers. The OpenID Connect core specification[31] provides a myriad of feature for authorization servers and clients to implement to make *"… simple things simple and complicated things possible"*[32]. It does not, however, make any guarantees of interoperability between a given client and server implementation. To ensure interoperability, implementations must have a minimum agreement on features each will implement.

[28] JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants - http://tools.ietf.org/html/draft-ietf-oauth-jwt-bearer-12#section-5
[29] JSON Web Algorithms (JWA) (Draft 40) - https://tools.ietf.org/html/draft-ietf-jose-json-web-algorithms-40
[30] http://openid.net/connect/
[31] http://openid.net/specs/openid-connect-core-1_0.html
[32] http://openid.net/connect/faq/

Identity Token Signatures and Verification

As part of the authorization grant process, a client receives an identity token. The OIDC specification indicates in section 2 that this token must be signed, but makes allowances for such signatures to be signed using the “none” algorithm where tokens are not returned directly from the authorization endpoint (i.e. the client is required to use an authorization code to retrieve their access token).

This creates two possibilities:

- **Option 1**: All OAuth clients must utilize the authorization code grant type to have this feature, even when they are a **public client** (i.e. a client that cannot protect client secrets). This means that the authorization server would need to allow authorization code grants to be used anonymously for such clients. The resulting identity token could not be used in other contexts, such as in a client/server application that wants to prove to its server component the user that had been authenticated via local orchestration of the authorization workflow.
- **Option 2**: Authorization servers would need to advertise public signing keys, via a pre-negotiated URL or via a discovery mechanism, used to sign identity tokens. Furthermore, the server must compute a hash value for the authorization token and include the value in the identity token to ensure that the user cannot co-mingle tokens obtained from the same authorization server via other sessions; clients must compute the same value and ensure the values match. Finally, clients would be required to perform signature validation, using keys fetched from the identity provider. Such an identity token could be reused with other services as proof of user authentication to a given client.

From a technical perspective, the first solution reduces the overall technical complexity required to implement OIDC for both clients and servers, while still achieving the same level of security, and is thus the recommended approach. For this reason, the **response_type** of “code” is recommended here as the minimum baseline for clients and authorization servers to interoperate.

Other Requirements to use OIDC

- Clients and servers must support section 3.1 “Authentication using the Authorization Code Flow” of the OIDC specification.
- Clients must establish a session with the user who is accessing the client application (even if not authenticated) and provide a “state” parameter in the authorization request that is tied to that session. This security mechanism prevents external actors from “pushing” authorization tokens that do not belong to the user onto a user’s client. An example is a malicious patient tricking another patient to enter sensitive data (such as insurance information) into the malicious patient’s account.
- Authorization servers must support request for the following claims as part of the id_token as required by OIDC:
    - **iss -** the issuer identifier of the authorization server.
    - **sub** - the unique identifier for the individual accessing the system.
    - **aud** - the client identifier of the intended recipient.
    - **exp** - the date/time of expiration of the token when used for authentication.

    - **Iat** - the date/time when the token was issued.
- Per the specification, authorization servers must support all other items denoted in section 15.1 of OpenID Connect Core Specification[33]. **Note**: As these servers are only required to support the token endpoint, signatures are not required, per the specification.

Privacy of Individuals and Identifiers

When conveying subject claims, authorization servers must consider the use of **Pairwise Pseudonymous Identifiers** per section 17.3 “Privacy Considerations” of the OIDC specification for individuals using clients to access their personal health details. This is to prevent correlation that could make their information personally-identifiable.

Session Management

The current draft for OpenID Connect session management[34] relies on iframe postMessages and makes assumptions on how and where session state is managed. This mechanism is incompatible with the use of native client applications, as sessions generally will not be stored in a browser directly whose state is directly accessible to the client application. Additionally, this specification does not cover automatic log-off[35] scenarios, which are generally required in healthcare venues for providers.

Session Management and Automatic Logoff

As noted above in the risks section, automatic logoff is an addressable requirement for HIPAA for users operating within a covered entity or as a business associate. This may impart requirements on the authentication and authorization systems in scenarios with shared devices that do not support the concept of user sessions.

Part of this approach would be to utilize refresh tokens; an application would utilize the fact of whether a refresh token can be obtained as a mechanism to determine if a user’s session is still active. The fact that an application requests a refresh token, however, cannot be assumed by an authorization server as an indicator that the user is currently active in a given client application. On many platforms, it is not possible for applications to detect whether a user has been active or coordinate with other applications; as such, it would be prudent for this requirement to be delegated to the operating environment itself.

[33] http://openid.net/specs/openid-connect-core-1_0.html#ServerMTI
[34] http://openid.net/specs/openid-connect-session-1_0.html
[35] HIPAA Security Series Technical Safeguards 164.312(a)(2)(iii) - http://www.hhs.gov/ocr/privacy/hipaa/administrative/securityrule/techsafeguards.pdf

Handling Errors in the Authorization Process

Errors can occur during different steps of the lifecycle of the authorization process. In the OAuth 2 specification, the following parameters may be in each of the possible response messages:

- **error** – A single ASCII value denoting the type of error.
- **error_description** – A human-readable description of the error, intended for the client developer.
- **error_uri** – The location of a human-readable web page describing the error, intended for the client developer.

As noted above, the self-describing mechanisms in the error handling process are intended for the developer of an application, not the user. As such, it becomes the responsibility of the authorization server to convey to the user the reason for authorization failures before returning the user to the client application with an error. The authorization server is also responsible for providing the additional information for the benefit of the client application to assist in any necessary troubleshooting.

During the initial authorization process, or while a client is operating, the authorization of a given user or a client application may be or become insufficient to fulfill a request. This can manifest itself during different steps of the OAuth2 workflow, which include:

- Obtaining an authorization code
- Obtaining an access token (including refresh tokens)
- Utilizing an access token

Reasons for authorization failure could include:

- The user declined the authorization request.
- The client is not registered.
- The client is no longer authorized to make any requests.
- The redirect URI supplied did not match one registered for the client.
- A user's account has been locked for security purposes.
- The user's session has ended or been otherwise terminated.
- The client's long-lived authorization token was revoked.
- The user does not currently have access to the resource.

During workflows that involve refreshing access tokens or utilizing access tokens, the authorization server does not have an opportunity to interact with the user to inform them to the cause of failure. Furthermore, with the information provided by the base OAuth 2 error message parameters, a client may be unable to differentiate between these specific reasons to provide users a reasonable course of action. To address this, it is recommended that authorization servers and resource servers provide an error_uri parameter in such responses whose content is meaningful to users and directly contains or links to technical information for support staff or developers. Client applications are responsible for directing users to the URI in a user agent for more information, where desired.

TLS / Encryption in Transit

To ensure data transport encryption considerations are up-to-date for participating system, the IETF RFC "Recommendations for Secure Use of TLS and DTLS[36]" is recommended as the minimum set of requirements for clients and servers to securely exchange all protocol messages. As a "best current practices" document, implementations should have a strategy for staying current with best security practices.

JavaScript Clients

To enable JavaScript-only clients, authorization servers can enable cross-origin resource sharing (CORS) for their access token endpoints. This will allow client-side JavaScript to exchange authorization code grants for access tokens using XMLHttpRequest.

Additionally, to reduce the complexity of such clients, it is prudent to allow clients to be registered using a client-defined URN as their client identifier rather than one dynamically assigned by the authorization server.

Functional Security Considerations

Required Administrator Functions

Administrative tooling for the authorization server will need to incorporate the following features to enable administrators to on-board new applications.

Registering Confidential Clients

To on-board a new client application, an administrator must have tooling to register the site's OAuth redirection URI and the location of their public key document. Such on-boarding should include choices as to what scopes the application will be capable of accessing. Additionally, the registration should allow the administrator to limit the maximum duration of access tokens issued to the client, or if tokens are bound to the lifecycle of a user's session.

Registering Native Applications / "Public Clients"

To on-board a new OAuth "public client", an administrator must have tooling to register the client's redirection URI. Similar to confidential clients, such public clients may be restricted in what scopes they can request.

Trust Store Management

The administrative tooling should allow for administrators to add and remove from a pre-configured trust store of public certificate authorities. Authorities that are considered trustworthy will change over time, and critical changes may occur outside of software updates delivered for the software. Additionally, some organizations may choose to reduce their risk posture by trusting fewer authorities than what is trusted by the general public.

[36] Recommendations for Secure Use of TLS and DTLS https://datatracker.ietf.org/doc/draft-ietf-uta-tls-bcp/

Contact Instructions
The administrative tooling should allow the administrator to craft instructions for users who access applications that have not been authorized. Such tooling may allow different messages for different classes of users (such as care recipients versus care providers).

Configuration of Entitled Users
Certain classes of users may require security or compliance training before being allowed to use third-party applications. Administrative tooling should provide the capability to lock confidential client or public client access to specific users.

User Authorization Required
Certain combinations of user types, security scopes, clients, or token durations may require explicit approval from the user. Administrative tools should provide reasonable configuration options for defining when user approval is required, with sensible defaults. The tooling should allow the administrator to customize messaging to user – in many situations, it may be more appropriate to convey the risk of such approvals in plain language instead of just a listing of what will be approved.

User Session and Grant Termination
In the event of a security incident, an administrator should have the capability of terminating all sessions for a given user or set of users. This termination, in turn, should revoke session-based authorization tokens. Additionally, the administrator should have the ability to review any long-lived grants that may have been issued on behalf of the user(s) and revoke those, as well.

Autonomous Access
Certain clients should be allowed to have direct access to services without user involvement (i.e. “B2B” services). The tooling should allow the administrator to configure which confidential clients are able to function using the client credentials grant type.

Pseudonymous Care Recipient Subject Identifiers
Administrative tools should allow the administrator to configure which systems will receive pairwise pseudonymous identifiers for a given user, rather than information that would allow direct correlation with other sites. This allows for some degree of anonymity with client applications targeted at consumers that do not need the individual’s demographic information to operate.

User-Facing Security Options

Terminate Previous Sessions
A user may log in using another device, having previously forgotten to log-out at another device. The authentication process should provide the ability to inform the user about other sessions and provide the option to terminate such sessions.

Terminate Grants
A user may wish to remove long-lived permissions granted to an application. Tools should be provided to the user to remove these grants, where desired, with friendly explanations to what the grants mean to their security.

Phishing Protection
User facing authentication solutions should incorporate means of phishing protection. For browser-based authentication, this may incorporate the use of additional security set-up to configure a persistent cookie. For shared devices, this might incorporate a similar function, pushed by the administrator at log-in or when the device is set up. Other possibilities include a native authenticator that must be invoked directly by the user before accessing applications.

Technical Security Considerations

OAuth Precautions

Use of OAuth State Parameter for CSRF Protection
Applications should establish a "session" with the user before the authorization process begins. This session should store an opaque state token that is associated with the user's session using a value other than the private secret used to identify the session. This value should be passed to the authorization server as the state parameter, and verified that the same value is returned with any authorization token or authorization code it receives. The token should be rejected if these values do not match, or if the user does not have an established session with the client. This mechanism is necessary to prevent the injection of authentication or authorization from external sources. Such attacks have implications in account-linking scenarios, online payment, or could result in information disclosure.

Verifying Audience of Authorization Token
If a client application protects resources other than those that are granted by the authorization grant, it must verify that it was the intended recipient of the resulting token. The audience ("aud") parameter served via the OpenID Connect id_token serves this purpose, and must be checked by client applications to prevent a second, unrelated client application from usurping the identity of the user.

Prevention of Click-Jacking Approval
Authorization servers should implement technologies to ensure that users cannot be tricked into erroneously approving an application via UI redress techniques[37].

Detecting "Counterfeit" Resource Servers
As clients may not have an explicit white-list of all resource servers that will interact with, an additional "aud" parameter should be sent with any OAuth request indicating a base URL for the resource server that the resulting access token will be utilized with. Authorization servers should store this information as part of the resulting token, and resource servers should reject any tokens received where this value does not match the base URL of the service(s). The contract between the authorization server and resource server for encoding this information is outside the scope of this document.

[37] https://www.owasp.org/index.php/Clickjacking

Isolation of Different "Classes" of Entities

Implementations should incorporate isolation of computational resources between classes of users (providers versus health plan members), isolate trusted versus untrusted systems, or throttle services based on information about the device, user, application, or location. This is to ensure the primary functions of an electronic health record can continue to function, even when an active denial-of-service is occurring (where such services have been made publically available). Furthermore, an implementation may wish to incorporate different authorization services for users within the internal network that are separate from those accessing it via public networks.

Usability Considerations

Discovery and "Launch" Mechanisms

EHR applications and clients may need ways to discover each another. Security mechanisms on native platforms and/or origin policy within browsers prevent simple means of this discovery from occurring (generally, as a matter of privacy). Such intra-application coordination can be highly complex (see CCOW[38] as an example) – as such, this section only discusses the security implications of how such coordination should occur.

Launching a Web Application from the EHR

When launching from the EHR, the EHR system can explicitly know the location of registered third party client applications and present links to them in-line with appropriate workflows within the record.

One issue that will be encountered is how to handle return visits to REST-based resources in a client application once a user is no longer authenticated. This could occur when a user shares a bookmark for the resource, or returns after browser storage has been cleared. This creates one of the following requirements:

- The REST URL will need to contain information that allows the client application to re-start the authorization process.
- A process to discover which EHR system the user wants to utilize exists.
- The user is informed to return to their EHR to access the solution (least optimal).

With the availability of permanent cookies / HTML5 storage, a client web application may track which EHR systems have previously launched it, allowing it to present the option to users upon return.

Launching a Native Application from the EHR

Launching a native application from the EHR is similar to launching web applications, with the notable difference that such launching links are generally not presented to the user. Additionally, most applications do not lose storage unless uninstalled and re-installed. As such, it may be justifiable to instruct a user to start the workflow from the EHR in the case of a fresh installation.

[38] HL7 CCOW - http://www.hl7.com.au/CCOW.htm

A given EHR will not have the ability to detect if the suitable native application is installed on the user's device, unless the EHR itself is a native application. As a web application, the EHR can at best attempt to launch a URI associated with the application, which will in turn cause the operating system to either open the application or prompt the user to search for a suitable application in an application store.

Embedded Web Views

Native applications should not leverage embedded web views for orchestrating OAuth authorization. Doing so can interfere with proprietary authentication mechanisms (including phishing protections) employed by an EHR system, prevents single-sign-on, and increases the risk of leakage of a user's credentials used for authentication. An EHR may choose to embed web views – it is assumed that a native application that is part of an EHR has a direct relationship with the authorization server and will appropriately handle any requirements the authorization server requires for security.

Summary of Recommended Implementation Requirements and Best Practices

The following section uses RFC terminology[39] to describe the required, recommended, and optional implementation requirements for this framework.

Authorization and Resource Servers

OAuth2 Support

- **MUST** support the OAuth 2 framework using the **Authorization Code Grant** workflow.
- **MUST** support issuance of the **Bearer** token type for use with services.
- **MUST** provide an error_uri parameter for any failure to obtain an authorization code, authorization token, or when rejecting an authorization token at a resource server.
    - **MUST** provide a web page, located at the error_uri location that is meaningful to both users and developers for describing why the failure occurred, and what steps to take.
    - **SHOULD** provide such information directly to users where the failure occurs when interacting through the authorization server.
- **IF** issuing authorization tokens that last longer than a user's session, or issuing tokens via a client credentials grant, the authorization server **MUST** require authentication for claiming/refreshing such access tokens.
    - **MUST** support the use of the JWT Bearer Token specification for this authentication, conformant to the section named "Interoperability Requirements" in the "Client Authentication" section of this document.
    - **SHOULD** take precautions to prevent excessive usage of a client's JSON Web Key document, such as utilizing cache headers returned by the client application.
- **IF** issuing authorizations that are valid for the length of a user's session or longer, the authorization server **SHOULD** support the use of refresh tokens.
    - **MUST** issue authorization tokens for short durations (generally, shorter than the life of a user's session)
    - **MUST** verify at the authorization server if the associated session or long-lived grant has been revoked when issuing access tokens from refresh tokens.
- **RECOMMENDED** to support the **Client Credentials Grant** workflow for remote services that need autonomous access.
- **SHOULD** allow clients with unique redirection URIs to use their redirection URI as a client identifier.
- **SHOULD** utilize scope names that have a low probability of conflicting with other OAuth-based specifications.
- **SHOULD** incorporate support for the special scope of "offline_access" for identifying a client's request to receive an infinitely-lived grant.

[39] RFC Terminology http://www.ietf.org/rfc/rfc3536.txt

OpenID Connect Support

- **MUST** support the "code" grant in conjunction with an OpenID Connect request.
- **MAY** support other grant modes.
- **MAY** support signatures of id tokens.
- **MAY** support other features, such as discovery.
- **RECOMMENDED** to support the creation of pairwise pseudonymous identifiers for user subjects as necessary to protect privacy.

TLS

- **MUST** utilize practices outlined in the IETF best current practice "Recommendations for Secure Use of TLS and DTLS" and should continue to evaluate conformance on a scheduled basis.
  - **MAY** implement changes that impair interoperability with clients where necessary for addressing newly discovered vulnerabilities that impact the confidentiality of services.

Security and Risk Mitigation

- **MUST** implement functions to prevent UI redress attacks that could cause a user to incorrectly approve an authorization.
- **SHOULD** implement features to prevent denial-of-service attacks that could affect the availability of critical functions of the electronic health record.
- **SHOULD** provide users the ability to terminate previous sessions.
- **SHOULD** provide users the ability to terminate access to previously approved clients.
- **RECOMMENDED** to provide mechanisms that allow an administrator to enforce the presence of antivirus or MDM solutions that protect against malware on a user's device, such as device white-listing.
- **RECOMMENDED** to provide phishing protection mechanisms to users during the authentication process where secrets entered by the user are used as a means of authentication.
- **RECOMMENDED** to capture a device's identifier during the OAuth 2 credentials grant workflow if a long-lived authorization is being issued to a public client application and provide tooling that allows a user or administrator to revoke all such authorizations associated with a given device.

Administrative Tools

- **MUST** provide the capability to register client applications.
  - **MUST** provide the ability to register a client's redirect URI.
  - **MUST** provide the ability to register a client's JWK URI (for use with client authentication).
  - **SHOULD** allow the use of a URN as defined by the client application as a client identifier.
  - **RECOMMENDED** to display authoritative information present in the SSL certificate for such endpoints to the administrator for use in identity proofing the organization providing the client application, where available.
- **MUST** provide the capability of managing the authorization server's trust store for retrieving the public keys of client applications.

- **SHOULD** provide the administrator the ability to configure custom messages for the user when an application is not authorized.
- **SHOULD** provide the administrator the ability to control which users are capable of utilizing specific client applications.
- **RECOMMENDED** that administrators are given fine grain control over when to prompt the user for explicit approval, along with configurable messages to describe the associated risk.
- **SHOULD** provide tooling for the administrator to terminate user sessions and long-lived grants that have been approved by users.
- **RECOMMENDED** that administrators may configure clients that utilize client credential grants (e.g. autonomous access, servers that utilize offline access tokens).
- **RECOMMENDED** that administrators are allowed to configure client applications to receive pseudonymous identifiers for care consumers instead of the normal user identifier.

Client Applications

OAuth2 Support

- **IF** utilizing workflows that are intended for use by an individual, clients **MUST** support the OAuth 2 framework using the **Authorization Code Grant** workflow to receive a bearer token.
    - **IF** utilizing authorizations issued for the lifetime of a user session or longer, clients **MUST** support the use of refresh tokens.
    - **MUST** verify the audience of the authorization token was intended for their client application, using the audience parameter contained in the id_token.
    - **MUST** utilize the “state” parameter to prevent malicious parties from substituting their own authorization on an unknowing user.
    - **MUST** invoke the user approval within a native browser (i.e. not embedded in the application).
    - **SHOULD** request the scope of “offline_access” when requesting a grant to be infinitely lived.
    - **SHOULD** present the error_uri as a link when an error occurs, allowing the user to obtain assistance/information as to the cause.
- **IF** acting as an autonomous service with no user interaction, clients **MUST** support the OAuth 2 framework using the **Client Credentials Grant**.
- **IF** required to authenticate, clients **MUST** do so using the JWT Bearer Token specification, conformant to the section named “Interoperability Requirements” in the “Client Authentication” section of this document.

OpenID Connect

- **IF** utilizing workflows that require identification of the current user, it is **RECOMMENDED** to utilize OpenID Connect.
    - **MUST** support the code workflow for obtaining an id_token.
    - **MUST** be capable of accepting unsigned id tokens obtained via code grant.

TLS

- **MUST** utilize practices outlined in the IETF best current practice “Recommendations for Secure Use of TLS and DTLS” and should continue to evaluate conformance on a scheduled basis.
    - **MAY** implement changes that impair interoperability with clients where necessary for addressing newly discovered vulnerabilities that impact the confidentiality of services.

Operational Best Practices

The following are security best practices that should be implemented by the operator of an interoperable EHR:

- **MUST** have established procedures for verifying the business identity of parties providing client applications.
    - **MAY** utilize third parties for identity verification, such as information collected and asserted by a certificate authority.
    - **MAY** utilize registration information maintained by other registries, such as one provided by an EHR vendor.
- **SHOULD** take measures to implement or assess “workstation security” requirements for devices used by care providers.
    - **RECOMMENDED** to implement measures to control end user devices for care providers, such as through the use of locked-down user profiles or a mobile device management solution. Such measures are useful in preventing malware and ensuring protecting unattended devices.
    - **MAY** implement procedures to perform periodic reviews of uncontrolled mobile devices in lieu of locked-down environments.
    - **MAY** implement mandatory user training programs for care providers utilizing uncontrolled mobile devices.