Argonaut Project:
Authorization Risk Assessment

Version 1.0 (Phase 1)
May 26, 2015

Argonaut Project:
Authorization Risk Assessment

1. Introduction

The Argonaut Project is a multi-stakeholder initiative to advance the development and adoption of open, API-based interoperability in the health care industry. Specifically, the Argonaut Project aims to produce specifications and implementation guidance that will enable software vendors and healthcare organizations to implement software applications capable of querying electronic health records (EHRs) for discrete data elements and structured documents using RESTful services specified using HL7 Fast Healthcare Interoperability Resources (FHIR).[1] The Argonaut Project commenced by defining four use cases,[2] selected as representing highly functional scenarios that present manageable security risks, and containing trust risks to within a single healthcare organization. These use cases are:

1. Patient uses provider-approved web application to access health data
2. Patient uses provider-approved mobile app to access health data
3. Clinician uses provider-approved web application to access health data
4. Clinician uses provider-approved mobile app to access health data

A critical factor in the success of the Argonaut Project is to assure that only authorized human users and software applications are able to access EHR data using these APIs, and that they are able to access only those resources for which they have been authorized. The OAuth 2.0 Authorization Framework[3], developed by the Internet Engineering Task Force (IETF), is well suited for this purpose and has been profiled by the open *SMART Platforms* initiative in several specifications.[4,5,6]

2. Purpose

The purpose of this risk assessment is to examine the *SMART on FHIR* authorization specifications with respect to how they address security risks associated with the use of OAuth 2.0, and the OpenID profile of OAuth 2.0, for the identified use cases, and to identify changes warranted for the Argonaut authorization profile(s).

3. Approach

3.1 Sources

The risk assessment was based on threats, vulnerabilities, and mitigation recommendations provided in the following specifications:

- The OAuth 2.0 Framework,[3] Section 10 (Security Considerations)
- The OAuth 2.0 Threat Model and Security Considerations,[7] Sections 4 (Threat Model) and 5 (Security Considerations)

- The OAuth 2.0 Authorization Framework: Bearer Token Usage,[8] Section 5 (Security Considerations)
- OpenID Connect Core 1.0,[9] Section 16 (Security Considerations)

In addition, a security analysis[10] performed by the MITRE Corporation for the Department of Veterans Affairs was reviewed.

3.2 Process

The risk assessment process comprised the following four steps:

1. Review the security risks identified in the sources above, and assess their applicability to the four use cases defined for the Argonaut Project.[11]
2. Examine the *SMART on FHIR* Authorization profiles with respect to the mitigations recommended for each applicable security risk, and identify gaps.
3. For each gap, assess the readiness of the recommended mitigations, considering the maturity and adoptability factors and metrics developed by the Health Information Technology Standards Committee.[12]
4. Coordinate recommendations for additional mitigations.

4. Findings

Specific results of the risk assessment are given in Appendix A. Each row provides:

1. Reference to the source documentation
2. Description of the risk
3. Summary of recommendations contained in the source documentation
4. Description of whether and how the *SMART on FHIR* authorization profiles[1] address the risk
5. Description of changes implemented in the Argonaut authorization profile

5. Argonaut Profile Modifications

At the time of this assessment, the *SMART on FHIR* authorization profiles included two profiles: a confidential profile, for apps capable of protecting a secret used for authenticating themselves; and a public profile, for apps that are not capable of protecting a secret. These profiles referenced a separate specification that provided details on scopes and launch context.

As the assessment proceeded, we noted a need for some high-level changes to enhance clarity and consistency with the OAuth 2.0 specifications, and to enhance implementability, with the ultimate objective of encouraging widespread adoption both within the Argonaut community and beyond. Section 5.1 descries these high-level modifications that were made to the *SMART on FHIR* profiles to create the single Argonaut profile.

[1] As of the start of the Argonaut Project.

Section 5.2 describes more granular, detailed changes that were made, directly traceable to the risk assessment, the results of which are provided in Appendix A.

5.1 Profile-Level Changes

5.1.1 Two profiles merged

At the time of this assessment, the *SMART on FHIR* authorization profiles included two profiles: a confidential profile, for apps capable of protecting a secret used for authenticating themselves; and a public profile, for apps that are not capable of protecting a secret. The two profiles were quite similar: both implemented the Authorization Code Grant model, which requires the client to obtain an authorization code that is then exchanged for an access token. The only real difference was that the confidential profile authenticated itself to the authorization server when it exchanged the code for an access token, and when it exchanged a refresh token for a new access token. So for simplicity in understanding and implementation, and ease of on-going profile maintenance, the Argonaut Project elected to merge the two profiles into a single profile, with differences in applicability between confidential and public clients clearly noted.

5.1.2 Separation of OAuth actors

The OAuth 2.0 profile identifies two central actors whose behaviors are central to the Argonaut use cases: the authorization server, which mediates applications' requests for access to resources and grants access consistent with organizational policy and end-user permissions, and the resource server, which holds the resources being accessed.

The *SMART on FHIR* profiles combined these two actors into one "EHR" actor, creating a possibility of misinterpretation for implementers. The Argonaut profile separately addresses these two actors as the "EHR authorization server" and the "EHR FHIR resource server."

5.1.3 Use of token to access resource

The *SMART on FHIR* profiles did not describe the final step – the app's use of the access token to access a resource. Although this step was shown in the sequence diagram, it was not described in the text.

The Argonaut profile adds a description of this final step – an app presenting a "bearer" token to the FHIR resource server. To counter the risk of token leakage, the app presents the token to the resource server within an Authorization header (see Appendix A).

5.1.4 Clarification of launch context and OAuth 2.0 execution sequences

The *SMART on FHIR* profiles began by describing the launch sequences for an app launching from within the context of an EHR, and as a stand-alone ("native") application. This description duplicated content that later appeared in the

authorization description, which made it somewhat challenging to interpret. In addition, each of the two profiles included a single sequence diagram.

For increased clarity, five new sequence diagrams were developed: 1) EHR launch; 2) standalone launch; 3) app request for authorization; 4) app exchange of authorization code for access token; and 5) app use of bearer token to access FHIR resources, and exchange of refresh token for new access token. The description of the app launch was separated from the specification of OAuth2 authorization request and use. Finally, a new, brief overview of the entire process was added.

5.1.5 User identity authentication

When an app desires to authenticate the identity of an end-user, it may do so by asking the authorization server for an OpenID Connect token attesting to the user's identity. The app sends the request to the authorization server by including two parameters in the scope of the authorization request: `openid` and `profile`. Although the *SMART on FHIR* profiles included this information in the scopes and launch context specification, requesting an ID token was not integrated into the profile descriptions or examples.

The Argonaut profile integrates the request for end-user identity authentication into the body of the profile, with the app receiving an ID token in response to the request.

5.2 Specific Changes

The following summarizes the specific modifications that were made in translating the *SMART on FHIR* authorization profiles into the Argonaut profile, as a result of this risk assessment.

5.2.1 End-user authorization

The *SMART on FHIR* profiles included asking for end-user authorization for an app to access EHR data as "optional." Although the decision of whether to ask for end-user authorization is a policy decision made by the authorization server, we added the general rule that if an EHR launches an app (confidential or public) for an authenticated user who has explicitly requested the launch, end-user authorization is "optional." Else the authorization server "should" request the user's authorization.

5.2.2 App responsibility to protect

Although only confidential apps have a provable ability to protect secret information, both confidential and public apps have a responsibility to implement measures to protect authorization codes, access tokens, ID tokens, and refresh tokens from unauthorized access, use, and modification. The Argonaut profile includes specific guidance for app developers to help them build apps that fulfill this obligation. For example, app developers are warned not to store bearer tokens in cookies that are transmitted in the clear, and to assure than any values the app receives are not injected with executable code (e.g., SQL). The app developer is also given guidance on how

an app should behave should it receive a FHIR resource containing a 'reference" to a resource hosted on a different server.

5.2.3 Refresh tokens

OAuth 2.0 enables an authorization server to take several actions in response to an app's request for authorization to access a protected resource. The authorization server can refuse the request, it can issue an access token only, or it can issue a refresh token along with the access token. An app can exchange a refresh token for a new access token, when an access token expires. *SMART on FHIR* profiles included only issuance of an access token (or denial of access), which tends to encourage issuance of tokens with long expiration times, which carries attendant risks, as the authorization server has no opportunities to retract the access token during the token's lifetime. Refresh tokens enable authorization servers to issue access tokens with shorter expiration times, along with a refresh token that can be exchanged for a new access token, giving the authorization server opportunities to refuse the exchange.

The Argonaut profile includes the issuance and exchange of refresh tokens, and recommends assigning shorter expiration times to access tokens, and longer to refresh tokens. Refresh tokens require the same protection as access tokens, both at rest and in motion. Confidential clients are required to authenticate themselves when exchanging a refresh token for a new access token.

To protect against the risk of a counterfeit resource server phishing for access tokens, the Argonaut profile includes an additional parameter in the app's request for authorization. The `aud` parameter passes the FHIR resource server's endpoint URI to the authorization server, along with the request for access. The authorization server then can validate that the URI is a known and trusted value prior to returning an authorization code.

5.2.4 "State" requirement

A primary OAuth 2.0 risk is cross-site request forgery (CSRF) in which an attacker causes the user agent (e.g., browser, mobile device) to follow a malicious URI instead of the app's authorized URI, resulting in the app's using the attacker's authorization code or access token to access the attacker's resource instead of the protected resources the app actually intended to access. The use of the `state` parameter to help ensure continuity of a client's "session" throughout the OAuth flow is a key protection against CSRF.

The *SMART on FHIR* profiles "recommended" the use of the `state` parameter, but did not require it. The Argonaut profile "requires" the use of the `state` parameter wherever it is valid in the OAuth flow. The app then can assure that any request sent to its redirection URI includes a `state` value binding the request to the user-agent's `state`; the authorization server can assure that any authorization request includes this `state` value; and can include his value when it redirects the user-agent back to the app.

5.2.5 "Aud" parameter

Access token phishing by a counterfeit resource server is another OAuth 2.0 risk was not addressed by the original *SMART on FHIR* profiles. In this threat scenario, an attacker pretends to be a resource server from which the app may want to retrieve a resource. A client sends a valid access token to the counterfeit resource server, which then uses that token to access other services and resources from the legitimate resource server, on the end-user's behalf.

A countermeasure for this attack is to have the app include in the request it sends to the authorization server, the endpoint URL of the resource server the app talked to. This is accomplished using the audience (aud) parameter. The authorization server then associates this endpoint URL with the access token it returns to the client. When the token is presented to the legitimate resource server, the resource server validates the association, enabling it to detect tokens presented by a counterfeit server.

The Argonaut authorization profile requires that the aud parameter be included in an app's request for authorization.

6. Next Steps

The Argonaut authorization profile is a living document that will be expanded and updated as use cases are added, and as additional threats and countermeasures are identified.

7. References

[1] Health Level 7. Fast Healthcare Interoperability Resources (FHIR). Available from http://www.hl7.org/implement/standards/FHIR-Develop/index.html (accessed 2/20/15)
[2] Argonaut Project. Use Cases for the Argonaut Project. DRAFT V0.2. Jan 18, 2015. Available from https://docs.google.com/document/d/1qKowbbUAXqNUUxd5lSgC7tXBDHvlzx5L3LF4fH87ecM/edit?pli=1. (accessed 2/20/15)
[3] Internet Engineering Task Force. RFC 6749. The OAuth 2.0 authorization framework. Oct 2012. Available from http://www.rfc-base.org/rfc-6749.html. (accessed 2/20/15)
[4] SMART Platforms. SMART on FHIR authorization: confidential clients. Available from http://docs.smartplatforms.org/authorization/confidential/. (accessed 2/20/15)
[5] SMART Platforms. SMART on FHIR authorization: public clients. Available from http://docs.smartplatforms.org/authorization/confidential/. (accessed 2/20/15)
[6] SMART Platforms. SMART on FHIR: scopes and launch context. Available from http://docs.smartplatforms.org/authorization/scopes-and-launch-context/. (accessed 2/20/15)
[7] Internet Engineering Task Force. RFC 6819. The OAuth 2.0 threat model and security considerations. Jan 2013. Available from https://tools.ietf.org/html/rfc6819. (accessed 2/24/15)
[8] Internet Engineering Task Force. RFC 6750. The OAuth 2.0 authorization framework: Bearer token usage. Oct 2012. Available from https://tools.ietf.org/html/rfc6750. (accessed 2/24/15)

[9] OpenID connect core 1.0 incorporating errata set 1. Available from http://openid.net/specs/openid-connect-core-1_0.html. (accessed 2/24/15)
[10] Russell, M. Secure RESTful interface security analysis and guidance. MITRE Corporation. 2014. Available from http://secure-restful-interface-profile.github.io/pages/. (accessed 2/24/15)
[11] Use cases for Argonaut Project. Draft Version 0.2. January 18, 2015.
[12] Baker D, J Perlin, J Halamka. Evaluating and classifying the readiness of technical specifications for national standardization. *JAMIA*. Available at http://jamia.oxfordjournals.org/content/jaminfo/early/2014/12/17/amiajnl-2014-002802.full.pdf. (accessed 2/24/15)

Appendix A:
Risk Assessment Results

	A	B	C	D	E	F	G
1	**Ref**	**Threat Description**	**Hosted Web App**	**Mobile App**	**Countermeasures Recommended by Specifications**	**SMART on FHIR (as of Argonaut Project start)**	**Changes Implemented in the Argonaut Authorization Profile**
2	N/A	SMART profiles have single "EHR" actor that embodies the responsibilities and actions of both OAuth2 actors (authorization server and resource server), creating the risk of misinterpretation of OAuth 2.0 framework specification and threat model, resulting in an insecure implementation	x	x	Specifications assign actions relating to the issuance of access tokens and refresh tokens to an "authorization server," and actions relating to the use of an access token to retrieve a resource to a "resource server."	Assigns both authorization-related actions and resource-retrieval actions to "EHR."	Clarified the actions of the OAuth 2.0 actors using the references “EHR Authorization Server” and “EHR Resource Server” (or "EHR FHIR Server."
3	RFC6749, 10.2	Client impersonation - a malicious client impersonates a legitimate client and obtains unauthozied access to protected resources.	x	x	* Client authentication * Require full redirection URI * Provide user information re client, request, scope, time to use in deciding whether to authorize the client	* Authenticates confidential clients * Requires registration of full redirect URI for all clients * Asking user to authorize client is "optional." * Providing user information re client, request, scope, time is not currently included in the SMART profiles	If an EHR launches the app (confidential or public) for an authenticated user who has explicitly requested the launch, asking for end user's authorization is "optional;" else the authorization server SHOULD request the user's authorization. (not an app API issue) **Include in best practices for service implementers:** Need to configure authorization server to ask for user authorization as required in the organization's security policy. If the authorization server asks for user authorization, provide the user information regarding client, request, scope, time.
4	RFC6749, 10.3, 10.5 OIDC16.4, 16.5 RFC6750, 5.3	Unauthorized disclosure and use of access tokens	x	x	* Protect tokens in transit (TLS) and at rest * Assure that access tokens cannot be generated, modified or guessed * Clients should request minimal scope; authz server should grant minimal scope * Apps should not store bearer tokens in cookies that are transmitted in the clear	* TLS protects tokens in transit * Authorization Code Flow assures that access tokens are sent only between the AS and the client, and between the client and the RS -- not between the AS and the user agent * Protection of tokens held by the client is not addressed in the SMART profiles	Added explanation of the need for apps to protect access tokens and refresh tokens they hold, including the warning not to store bearer tokens in cookies that are transmitted in the clear.
5	RFC6749, 10.4 OIDC16.4, 16.5	Unauthorized disclosure and use of refresh tokens	x	x	* Protect tokens in transit (TLS) and at rest * Bind tokens to client-id whenever identity can be authenticated * Assure that refresh tokens cannot be generated, modified or guessed	Refresh tokens are not used. However, plan to add.	* Added refresh tokens to enable an app to request a new access token upon expiration. This enables access tokens to be issued with shorter lifetimes, and renewed as needed, and per policy. * Protect refresh tokens in transit (TLS) and at rest. * Added binding of tokens to client-id whenever client identity can be authenticated **Include in best practices for service implementers**: * Guidance regarding the need to assure that refresh tokens cannot be generated, modified or guessed.

	A	B	C	D	E	F	G
1	**Ref**	**Threat Description**	**Hosted Web App**	**Mobile App**	**Countermeasures Recommended by Specifications**	**SMART on FHIR (as of Argonaut Project start)**	**Changes Implemented in the Argonaut Authorization Profile**
6	RFC6749, 10.5	Unauthorized disclosure and use of authorization codes	x	x	* Transmission over TLS channel * Authorization codes should be short-lived and single-use * Authenticate client if possible	* Uses Authorization Code Grant model for both confidential and public clients * Transmission over TLS channel * Authorization codes are short-lived and single-use * Authenticates confidential clients	Added advice recommending lifetime of authorization codes (e.g., 1 minute)
7	RFC6749, 10.12	Cross-site request forgery (CSRF) -- attacker causes the user-agent to follow a malicious URI; client then uses the attacker's authorization code or access token to access the attacker's resources	x	x	* Use state parameter to ensure continuity of client "session" throughout OAuth flow: Client assures that any request sent to its redirection URI includes a value that binds the request to the user-agent's authenticated state (e.g., a hash of the session cookie used to authenticate the user-agent); client uses "state" request parameter to deliver this value to the AS when making an authorization request; once authorization has been obtained, AS redirects user-agent back to the client with the required binding value contained in the "state" parameter. * CSRF protection for AS end-point	Use of state parameter is recommended to assure continuiy of client session throughout OAuth flow	* Require "state" everywhere it is valid in OAuth flow (although we can't prevent apps from generating predictable state values, we can at least recommend a level of entropy). Client assures that any request sent to its redirection URI includes a value that binds the request to the user-agent's authenticated state (e.g., a hash of the session cookie used to authenticate the user-agent); client uses "state" request parameter to deliver this value to the AS when making an authorization request; once authorization has been obtained, AS redirects user-agent back to the client with the required binding value contained in the "state" parameter. **Include in best practices for service implementers:** Explicitly recommend CSRF protection for the AS endpoint -- a common oversight.
8	RFC6749, 10.14	Code injection -- when an input or other external variable is used by an application unsanitized and causes modification to the application logic	x	x	AS and client MUST sanitize (and validate when possible) any value received -- in particular, the value of the "state" and "redirect_uri" parameters	Not addressed. Client should make sure that values received are not injected with executable code.	* Added warning that AS and client should ensure that values received are not injected with executable code (e.g., SQL). Inputs should never be treated as executable code (e.g. they should never be slotted into SQL queries without escaping). * Clients must not forward values passed back to their redirect URLs to other arbitrary or user-provided URL.
9	RFC6819, 4.1.2	Attacker obtains refresh token from web app	x		* Standard web-server protections. * Strong client authentication (assertion/token).	Refresh tokens are not included in the profiles.	* Added refresh tokens. * Confidential clients should authenticate themselves to the AS when exchanging refresh token for access token
10	RFC6819, 4.1.2	Attacker obtains refresh token from mobile app		x	Store secrets in secure storage; lock device.	Refresh tokens are not included in the profiles	* Added refresh tokens. * Added mitigations to protect long-lived access tokens and refresh tokens (lock device; keep tokens in app-specific storage locations only, not in system-wide-discoverable locations)

	A	B	C	D	E	F	G
1	**Ref**	**Threat Description**	**Hosted Web App**	**Mobile App**	**Countermeasures Recommended by Specifications**	**SMART on FHIR (as of Argonaut Project start)**	**Changes Implemented in the Argonaut Authorization Profile**
11	RFC6819, 4.1.2	Obtain refresh token from stolen device		x	Lock device; have refresh tokens revoked.	Refresh tokens are not included in the profiles	* Added refresh tokens. **Include in FAQ for end users: Lock device.** **Include in FAQ for AS implementers:** Revoke access tokens by refusing to refresh when refresh token is presented.
12	RFC6819, 4.2.2	User unintentionally grants too much access scope				No recommendation for client to limit grants, scope, lifetime	Recommended that client limit grants, scope, lifetime, and that EHRs enable more granular access grants than "all." (See https://groups.google.com/forum/#!topic/smart-on-
13	RFC6819, 4.3.5	Obtaining Client Secret by Online Guessing	x		* Use high entropy for secrets * Lock accounts * Strong client authentication	AS authenticates confidential clients	Recommended high entropy for secrets, and limiting attempts + account locking when limit is exceeded.
14	RFC6819, 4.4.1.3	Online Guessing of Authorization "codes"	x	x	* Sign assertion-based tokens * Authenticate client * Bind code to redirect URI * Use short expiry times	* AS authenticates confidential clients * Authorization code bound to (validated) redirect URI * Expiry time is based on AS/organizational policy, which is outside the scope of these profiles	Recommended short expiry times for authorization codes. **Include in FAQ for AS implementers:** limit number of guesses.
15	RFC6819, 4.6.1	Eavesdropping Access Tokens on Transport between client and RS	x	x	* TLS channel between client and RS * Bind token to client ID * Short token lifetime	* Profiles do not include exchange between client and RS * Establishes TLS channel between RS and client * Tokens are bound to client ID * Token lifetime is determined by AS policy	* Added exchange between client and RS for resource retrieval. **Include in FAQ for AS implementers:** * Lifetimes for access tokens should be shorter than for refresh tokens (e.g., 1 hour vs. 1 year). * Lifetimes for access tokens for confidential clients may be longer than for public clients.
16	RFC6819, 4.6.2	Replay of Authorized RS Requests	x	x	TLS channel between RS and client	* Profiles do not include exchange between client and RS * Establishes TLS channel between RS and client	Added secure exchange between client and RS for resource retrieval.
17	RFC6819, 4.6.4	Access Token Phishing by Counterfeit Resource Server	x	x	* TLS to enable client to authenticate RS * Have client pass the RS endpoint URL along with the request, and then have the AS associate that RS endpoint with the access token returned, & the RS validate the association. * Client signs request to RS	* Profiles do not include exchange between client and RS, although shown in sequence diagram. * Client queries RS for URL to applicable AS.	* Added exchange between client and RS for resource retrieval. * Have the client pass the RS endpoint URL (via "aud" parameter) to the AS, along with the request for access. **Include in FAQ for AS implementers:** Have the AS validate that the RS URL is a known and trusted value prior to returning the authorization code.

	A	B	C	D	E	F	G
1	**Ref**	**Threat Description**	**Hosted Web App**	**Mobile App**	**Countermeasures Recommended by Specifications**	**SMART on FHIR (as of Argonaut Project start)**	**Changes Implemented in the Argonaut Authorization Profile**
18	RFC6819, 4.6.5	Abuse of Token by Legitimate Resource Server or Client -- one RS uses token to access resource in another RS, or client uses token to access resource in a different RS	x	x	FHIR data referencing a resource hosted on a different RS	Profiles do not include exchange between client and RS, although shown in sequence diagram	* Added exchange between client and RS for resource retrieval. * Clarified expectations about how a client should behave when it sees FHIR data with a "reference" to a resource hosted on a different RS. If a client blindly follows such references and sends along its access_token, the token is subject to potential theft, assuming an attacker can control the content of (some) resources on the RS.
19	RFC6819, 4.6.6	Leak of Confidential Data in HTTP Proxies (e.g., cache)	x	x	* Use OAuth HTTP authentication scheme to authenticate exchanges between client and RS; else use cache-control headers * Reduce scope and expiry times of access tokens	Profiles do not include exchange between client and RS, although shown in sequence diagram	* Added exchange between client and RS for resource retrieval. * Locked down the access token type to "Bearer," and to say these tokens are presented by the client to the RS in an Authorization header -- which is one of the proposed mitigations for the threat by which confidential data may leak to HTTP proxies (e.g., cache).
20	RFC6819, 4.6.7	Token Leakage via Log Files and HTTP Referrers (If access tokens are sent from client to RS via URI query parameters, such tokens may leak to log files and the HTTP "referer")	?	?	Use Authorization headers or POST parameters instead of URI request parameters when sending requests to RS Require authentication of requests sent to RS	Profiles do not include exchange between client and RS, although shown in sequence diagram	* Added exchange between client and RS for resource retrieval. * Locked down the access token type to "Bearer," and to say these tokens are presented by the client to the RS in an Authorization header -- which is one of the proposed mitigations for this threat.
21	OIDC, 16.1	OID request disclosure	x	x	(in addition to RFC6819) use authorization request parameter that enables request to be sent as signed, and optionally encrypted, JWT	* OID request and token handling are not integrated into the primary profiles, but are included in the scopes document. * Authorization requests are not sent as signed, and optionally encrypted, JWTs	* Integrated OID into body of confidential and public profiles.
22	OIDC, 16.2	OID server masquerading	x	x	Client authenticates OID server	* OID request and token handling are not integrated into the primary profiles, but are included in the scopes document. * Client authenticates AS/OP	Integrated OID into body of confidential and public profiles.

	A	B	C	D	E	F	G
1	**Ref**	**Threat Description**	**Hosted Web App**	**Mobile App**	**Countermeasures Recommended by Specifications**	**SMART on FHIR (as of Argonaut Project start)**	**Changes Implemented in the Argonaut Authorization Profile**
23	OIDC, 16.3	ID Token manufacture or modification	x	x	* Have OpenID provider digitally sign token * Transmit over secured channel	* ID token is digitally signed JWT * Transmitted over TLS channel	Received proposal to make JWTs un-signed => TLS authenticates the server and encrypts the link. It does not provide non-repudiation for the entity issuing the ID Token. ID token should continue to be digitally signed JWT transmitted over the TLS link. May be particularly important for exchanges between organizations.
24	OIDC, 16.7	Request repudiation	x	x	Client digitally signs request using key that supports non-repudiation; AS validates signature	Client does not digitally sign request	Considered adding that the client SHOULD (or MAY?) digitally sign the access request, with AS validating the signature. Decided not to add at this time.