H. Roßnagel, C. H. Schunck, S. Mödersheim (Hrsg.): Open Identity Summit 2021,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2017 71

FAPI 2.0: A High-Security Profile for OAuth and OpenID Connect

Daniel Fett[1]

Abstract: A growing number of APIs, from the financial, health and other sectors, give access to highly sensitive data and resources. With the Financial-grade API (FAPI) Security Profile, the OpenID Foundation has created an interoperable and secure standard to protect such APIs.

The first version of FAPI has recently become an official standard and has already been adopted by large ecosystems, such as OpenBanking UK. Meanwhile, the OpenID Foundation's FAPI Working Group has started the work on a the second version of FAPI, putting a focus on robust interoperability, simplicity, a more structured approach to security, and improved non-repudiation.

In this paper, we give an overview of the FAPI profiles, discuss the learnings from practice that influence the development of the latest version of FAPI, and show how formal security analysis helps to shape security decisions.

Keywords: Authorization; Authentication; Security; Interoperability

1 Introduction

Multi-banking apps and other fintech offerings usually need to access their users' bank accounts in order to retrieve account information and initiate payments. For many years, screen-scaping has been the norm in this area: Apps use their users' online banking credentials to access (directly or via a server) the bank's online banking website through an emulated browser and extract relevant information.

This approach is not only brittle, as it can break easily when elements in the online banking interface change, it is also dangerous: Fintech services get unlimited access to accounts and misuse of the credentials cannot be detected or prevented effectively. From the bank's perspective, this kind of access is largely indistinguishable from phishing attacks.

With PSD2 [Co15] in the European Union and similar efforts in other markets, APIs for account information and payment initiation have been introduced. Nonetheless, some operating modes in these APIs still require the user to enter their credentials into the app or service they are about to use. Besides carrying a similar potential for misuse of credentials as screen-scraping, this trains users that entering their online banking credentials on third-party apps and websites is harmless.

[1] yes.com AG, Hafenstrasse 2, 8853 Lachen, Switzerland, mail@danielfett.de

A token-based approach to delegation of authorization, such as the one offered by OAuth 2.0 [Ha12], can ensure that users can give apps and services access to their resources in a controlled and secure way.[2] For open banking use cases such as the one described before, it is, however, not sufficient to just prescribe the use of OAuth 2.0: The IETF standard only defines a *framework* for protocols but leaves a wide range of options in almost all areas of the communication. For a practical use in any larger ecosystem, a *profile* of OAuth 2.0 needs to be defined to ensure interoperability between participants in the ecosystem, to ensure an andequate level of security, and to define a common feature set.

To this end, the OpenID Foundation launched an effort in 2016 to create the *Financial-grade API Security Profile* (FAPI). While initially focussed on financial applications, the standard is agnostic towards the resources to protect and can be used for all kinds of APIs bearing a high inherent risk. Therefore, it has been adopted not only by Open Banking UK[3] and the yes.com open banking scheme[4], but also by the Consumer Data Right initiative in Australia[5]; an integration in the Norwegian health sector is ongoing.

In this paper, we give a brief overview of the development of and differences between FAPI 1.0 and FAPI 2.0 (Section 2). In Section 3, we introduce the security model underlying FAPI 2.0 and show how it was shaped by formal security analysis. In Section 4 we present the main features of FAPI 2.0 and discuss learnings from practice that have influenced the development. We conclude in Section 5.

2 FAPI 1.0 and FAPI 2.0: Overview

FAPI was created by the OpenID Foundation's FAPI working group in order to address the following challenges:

- **Security:** In financial applications, a high level of security must be guaranteed. On its own and without applying further restrictions, OAuth 2.0 is not suitable for high-security applications, as it provides several options that are only secure under moderately strong attacker models [FHK19, FKS16]. FAPI ensures that specific options are selected and extensions are applied which ensure that the resulting protocol is suitable for high-security applications.

[2] With OAuth 2.0, a user can give a *client* (app or website) access to their resources at a so-called *resource server*, e.g., a banking API. To this end, the client first redirects the user to an endpoint at the banks *authorization server*, where, in direct communication between the user's browser and the authorization server, the user authenticates to the bank using their regular online banking credentials. The user can then authorize the access by the client, which can be limited in terms of resources that can be accessed, actions that can be performed, and lifetime of the authorizaton. The user's browser is then redirected back to the client, carrying an *authorization code* in the URL. The client exchanges this code for an *access token* in direct communication with the authorization server's *token endpoint*. With the access token, the client can access the user's resources (e.g., retrieve recent transactions or initiate a payment) at the resource server.

[3] `https://standards.openbanking.org.uk/`

[4] `https://yes.com/docs`

[5] `https://consumerdatastandardsaustralia.github.io/standards/`

- **Interoperability:** FAPI standardizes the protocol features that are used by clients and servers in order to ensure an interoperability between all participants in an ecosystem. This comprises standard OAuth 2.0 features from RFC6749 [Ha12] and RFC6750 [JH12] as well as extensions, such as client authentication via mutual TLS (RFC8705 [Ca20]).
- **Common feature set:** FAPI further ensures that protocol features are available that solve commonly encountered problems in financial ecosystems, for example, the transfer of complex and fine-grained information on the authorization requested by the client. This prevents the creation of individual solutions that lack standardization and support by software vendors.

FAPI 1.0 is defined in two separate profiles, *Baseline* and *Advanced*. The Baseline profile [SJ21a], initially called *read-only*, provides a level of security that is considered sufficient for less-critical operations, such as viewing basic account information. The Advanced profile [SJ21b], initially called *read-write*, aims to provide a high level of security. At the same time, it provides basic features for message-level non-repudiation. FAPI 1.0 has become an official OpenID Foundation standard in March 2021.

The work on FAPI 2.0 has commenced in 2019. Compared to its predecessor, FAPI 2.0 aims to further improve interoperability between FAPI implementations by reducing optionality within the protocol, to provide a broader scope of features, to simplify development for implementers, and to create a well-defined security model. FAPI 2.0 again defines two security levels, but FAPI 2.0 Baseline [Fe21c] provides a higher security level than FAPI 1.0 Advanced with a broader scope of features and easier development. FAPI 2.0 Advanced [Fe21a] additionally provides full non-repudiation, i.e., all relevant protocol messages are signed (see Figure 1). A security model for FAPI 2.0 is defined in a separate document, the *FAPI 2.0 Attacker Model* [Fe21b]. The Baseline profile and the Attacker Model are on track to become so-called *Implementers Drafts* soon.

Medium Security, no non-repudation	FAPI 1.0 Baseline	
High Security, limited non-repudiation	FAPI 1.0 Advanced	FAPI 2.0 Baseline
High Security, full non-repudiation		FAPI 2.0 Advanced

Fig. 1: Rough comparison of security and non-repudiation levels in FAPI versions.

3 Security Model

Through a number of measures, all FAPI profiles achieve a higher level of security than plain OAuth 2.0 can offer.

First of all, less-secure protocol options are forbidden. The prime example is the OAuth Implicit Grant, where an access token is transferred through the user's web browser. This

mode of operation is forbidden in FAPI, as access tokens are susceptible to a number of attacks in the browser and during transfer [FKS16, Lo20]. As another example, redirect URIs in FAPI must be HTTPS URIs and have to be pre-registered with the authorization server, preventing common vulnerabilities with OAuth redirections [Lo20]. Finally, a major difference to many existing OAuth deployments is also that symmetric client secrets are disallowed in all FAPI profiles except for FAPI 1.0 Baseline.

The second security measure in FAPI is the mandatory use of a selected set of extensions improving the security of OAuth, such as *Proof Key for Code Exchange* (PKCE) [SBA15], which protects authorization codes from misuse.

3.1 Lessons Learned from FAPI 1.0

For FAPI 1.0, the selection of security measures was made on a case-by-case basis, based on commonly known best practices and specific attack scenarios.

In previous work [FHK19], we analyzed the security of the then-current draft of FAPI 1.0 in order to analyze whether this process yielded a secure OAuth profile or not. The analysis was based on the *Web Infrastructure Model* (WIM), the most comprehensive formal model of web servers, browsers, and other components of the web to date [Fe18].

For a formal security analysis in general, one or more sets of capabilities of attackers need to be defined, the *attacker model*. The security of (a model of) the protocol can then either be proven against this attacker model, or the analysis reveals an attack on the protocol. Commonly assumed capabilities of attackers in the analysis of network and web protocols are, for example: (1) the capability to participate in the protocol just as a normal user can, (2) the capability to operate an arbitrary number of endpoints (e.g., web servers), and (3) the capability to read and manipulate arbitrary (unprotected) network traffic (the *network attacker* model).

In [FKS16] and [FKS17], we showed security of OAuth 2.0 and OpenID Connect under a network attacker model. For FAPI 1.0, however, a much stronger attacker model was derived from the description of the profile, adding a number of critical attacker capabilities: (3) reading the contents of the authorization request and response, e.g., through a misdirected app-to-app communication on a mobile device, (4) reading the access token, e.g., due to phishing, and (5) controlling the token enpoint, e.g., because it was misconfigured [FHK19].

Parts of this attacker model were explicitly mentioned in the FAPI draft (e.g., phishing), other parts were hinted at by the selection of security measures. For example, hashing the PKCE verifier to acquire the PKCE challenge implies that capability (3) is assumed. With this very strong attacker model, our analysis yielded a number of attacks that were taken into account in later versions of FAPI 1.0. With mitigations in place, the security of FAPI 1.0 was proven in the formal model.

However, the analysis also revealed that reverse-engineering an attacker model from a specification is a complex process. It futher showed that an underdefined attacker model can introduce inconsistencies into the specification that can lead to false expectations regarding the security of a protocol. For example, while a hashed PKCE verifier in the authorization request implies a protection against an attacker with capability (3), the *state* value is not protected and an attacker with capability (3) can potentially mount a cross-site request forgery attack against the client.

Therefore, for FAPI 2.0, one goal is to explicitly define a consistent security model. Such a model not only informs security decision in the design process, it also clears the way for easier and more accurate security analyses. Finally, it helps to communicate the precise level of security the specification aims to achieve and can be used by implementers to identify areas where further security measures might be required.

Additionally, some of the security measures in FAPI 1.0 are based on existing OpenID Connect features, necessitating the use of OpenID Connect even if authentication or the transfer of end-user information is not desired and OAuth alone would be sufficient. This can create additional complexity for developers. Another goal for FAPI 2.0 is therefore to make OpenID Connect a fully optional part of the specification and ensuring security with native OAuth features and extensions.

3.2 Security Model of FAPI 2.0

The first step in the development of FAPI 2.0 was to create a security model [Fe21b]. It is based on the FAPI 1.0 attacker model identified in [FHK19] described above. It further extends the model by capabilities for an attacker to read requests to and responses from the resource server, and to tamper with responses from the resource server. This was motivated by the fact that these requests often go through reverse proxies that can be manipulated or write log files to less-protected environments.

FAPI 2.0 Baseline and Advanced use the same attacker model, i.e., both must ensure the security and integrity of the authorization (and possibly, authentication) process in the presence of attackers with the described capabilities. Appendix A shows the attacker model for FAPI 2.0.

For Advanced, additional non-repudiation goals are defined: it must be ensured that receivers of protocol messages can prove the origin and integrity of all relevant messages received.

In parallel to the development of FAPI 1.0, the IETF has worked on a document that describes current best practices for OAuth 2.0 deployments [Lo20]. Another goal of FAPI 2.0 is to align its recommendations with those from the IETF.

4 Core Elements of FAPI 2.0 Baseline

In the following, we briefly present the main elements that make up FAPI 2.0 Baseline according to the current draft. For the details, refer to [Fe21c]. The Advanced profile is still in early development, in particular regarding the non-repudiation of messages to and from the resource server, which likely requires the development of a new message signing scheme.

FAPI 2.0 is based on OAuth 2.0 as defined in RFC6749 [Ha12] and RFC6750 [JH12]. OAuth may be used in one of two modes, the *Authorization Code Grant* and the *Client Credentials Grant*, where a client accesses a resource on its own behalf. OpenID Connect [Sa] can be used to optionally provide information about the end-user (authentication).

Clients need to authenticate themselves to the authorization server using asymmetric methods, either TLS client authentication as defined in RFC8705 [Ca20] or using a JSON Web Token (JWT) signed using a private key, as defined in OpenID Connect [Sa].

Authorization codes are protected using PKCE [SBA15] and access tokens need to be sender-constrained, i.e., must be bound to a private key only known to the client. This can be achieved by binding the access tokens to a private TLS certificate key as defined in RFC8705 [Ca20] or via Demonstrating of Proof-of-Posession at the Application Layer (DPoP) [Fe20].

To mitigate attacks where an endpoint is misconfigured, as mentioned above, OAuth Server Metadata from RFC8414 [JSB18] must be used. With this extension, authorization servers publish their URLs and other meta information on a predefined URL. This enables an automatic discovery and configuration of the authorization, token, and other endpoints.

To prevent Mix-Up attacks [FKS16], the Authorization Server Issuer Identifier in Authorization Response extension [MzSF21] is used.

Traditionally, OAuth authorization requests contain all information in the parameters of a single URL to which the user's browser is redirected. The information that is transferred includes the kind of access the client requests (expressed as a list of strings in the *scope* parameter) and various security-related parameters. This approach bears three problems: First, the data is neither encrypted nor integrity protected, i.e., the user or malicious code running in the user's browser can read and/or modify the scope or security-critical parameters. Second, there is an upper limit to the length of the data, imposed by the maximum length of a URL. And finally, a list of strings is not expressive enough for complex use cases. For example, a client may want to acquire authorization to initiate a payment of a certain amount of money to a specific bank account using a defined reference text. Custom encodings can be used to circumvent this limit, but these are non-standard and often lack support by software vendors.

To solve these problems, two OAuth extensions where created in the IETF and have become part of FAPI 2.0:

Pushed Authorization Requests (PAR) give the client an option to send all data that is normally contained in the authorization request to a special PAR endpoint at the authorization server, using a POST request. Since the data is transferred via a direct, authenticated channel between the client and the authorization server, integrity and confidentiality of all parameters can be ensured. The client receives a unique identifier in response, which can be used in place of the original parameters in the authorization request.

Rich Authorization Requests (RAR) are enabled through a new authorization request parameter, `authorization_details`, with a pre-defined, extensible structure. It can be used to transfer complex authorization requirements. While the precise contents of the parameter depend on the ecosysteme or scheme it is used in, its fixed structure enables easier software vendor support.

Due to its benefits for security, PAR must be used in FAPI 2.0. RAR is to be used when the scope parameter is not sufficient to solve the use case at hand.

5 Conclusion

The work of the FAPI working group fosters the use of OAuth 2.0 and OpenID Connect in large ecosystems. To further support this, the OpenID Foundation has created a comprehensive Conformance Testing Program [Op], which helps to ensure that implementations actually comply to the FAPI specificiations.

FAPI 1.0 is in use today, is fully supported by the Conformance Testing Program, and has been shown to be a secure standard.

FAPI 2.0 targets an even higher degree of interoperability by providing a further reduced set of options and covering a broader set of use cases by standard features. FAPI 2.0 uses a new approach to ensuring the security of the protocol by underpinning security-related decisions with an attacker model. This model will also be the basis for future formal analysis efforts to ensure the security of FAPI 2.0.

The FAPI 2.0 Baseline and Advanced profiles are under development at the OpenID Foundation and the working group welcomes feedback from external entities.

Bibliography

[Ca20] Campbell, Brian; Bradley, John; Sakimura, Nat; Lodderstedt, Torsten: OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens. (RFC 8705), February 2020. `https://rfc-editor.org/rfc/rfc8705.txt`.

[Co15] Directive (EU) 2015/2366 of the European Parliament and of the Council of 25 November 2015 on payment services in the internal market, amending Directives 2002/65/EC, 2009/110/EC and 2013/36/EU and Regulation (EU) No 1093/2010, and repealing Directive 2007/64/EC, https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:02015L2366-20151223.

[Fe18] Fett, Daniel: An Expressive Formal Model of the Web Infrastructure. PhD thesis, 2018. https://publ.sec.uni-stuttgart.de/fett-phdthesis-2018.pdf.

[Fe20] Fett, Daniel; Campbell, Brian; Bradley, John; Lodderstedt, Torsten; Jones, Mike; Waite, David: OAuth 2.0 Demonstrating Proof-of-Possession at the Application Layer (DPoP). (draft-ietf-oauth-security-topics-16), 2020. https://tools.ietf.org/html/draft-ietf-oauth-dpop.

[Fe21a] FAPI 2.0 Advanced Profile (Draft), https://bitbucket.org/openid/fapi/src/master/FAPI_2_0_Advanced_Profile.md.

[Fe21b] FAPI 2.0 Attacker Model (Draft), https://bitbucket.org/openid/fapi/src/master/FAPI_2_0_Attacker_Model.md.

[Fe21c] FAPI 2.0 Baseline Profile (Draft), https://bitbucket.org/openid/fapi/src/master/FAPI_2_0_Baseline_Profile.md.

[FHK19] Fett, Daniel; Hosseyni, Pedram; Küsters, Ralf: An Extensive Formal Security Analysis of the OpenID Financial-grade API. In: 2019 IEEE Symposium on Security and Privacy (S&P 2019). volume 1. IEEE Computer Society, pp. 1054–1072, May 2019. https://publ.sec.uni-stuttgart.de/fetthosseynikuesters-fapi-sp-2019.pdf.

[FKS16] Fett, Daniel; Küsters, Ralf; Schmitz, Guido: A Comprehensive Formal Security Analysis of OAuth 2.0. In: Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security (CCS 2016). ACM, pp. 1204–1215, 2016. https://publ.sec.uni-stuttgart.de/fettkuestersschmitz-ccs-2016.pdf.

[FKS17] Fett, Daniel; Küsters, Ralf; Schmitz, Guido: The Web SSO Standard OpenID Connect: In-Depth Formal Security Analysis and Security Guidelines. In: IEEE 30th Computer Security Foundations Symposium (CSF 2017). IEEE Computer Society, pp. 189–202, 2017. https://publ.sec.uni-stuttgart.de/fettkuestersschmitz-csf-2017.pdf.

[Ha12] Hardt, Dick: The OAuth 2.0 Authorization Framework. (RFC 6749), October 2012. https://rfc-editor.org/rfc/rfc6749.txt.

[JH12] Jones, Michael; Hardt, Dick: The OAuth 2.0 Authorization Framework: Bearer Token Usage. (RFC 6750), October 2012. https://rfc-editor.org/rfc/rfc6750.txt.

[JSB18] Jones, Michael; Sakimura, Nat; Bradley, John: OAuth 2.0 Authorization Server Metadata. (RFC 8414), June 2018. https://rfc-editor.org/rfc/rfc8414.txt.

[Lo20] Lodderstedt, Torsten; Bradley, John; Labunets, Andrey; Fett, Daniel: OAuth 2.0 Security Best Current Practice. (draft-ietf-oauth-security-topics-16), 2020. https://tools.ietf.org/html/draft-ietf-oauth-security-topics.

[MzSF21] Meyer zu Selhausen, Karsten; Fett, Daniel: OAuth 2.0 Authorization Server Issuer Identifier in Authorization Response. (draft-ietf-oauth-iss-auth-resp-00), 2021. https://tools.ietf.org/html/draft-ietf-oauth-iss-auth-resp.

[Op] Conformance Testing for FAPI Read/Write OPs, https://openid.net/certification/fapi_op_testing/.

[Sa] Sakimura, Nat; Bradley, John; Jones, Mike; de Medeiros, Breno; Mortimore, Chuck: OpenID Connect Core 1.0 incorporating errata set 1. http://openid.net/specs/openid-connect-core-1_0.html.

[SBA15] Sakimura, Nat; Bradley, John; Agarwal, Naveen: Proof Key for Code Exchange by OAuth Public Clients. (RFC 7636), September 2015. https://rfc-editor.org/rfc/rfc7636.txt.

[SJ21a] Financial-grade API Security Profile 1.0 - Part 1: Baseline, https://bitbucket.org/openid/fapi/src/master/Financial_API_WD_001.md.

[SJ21b] Financial-grade API Security Profile 1.0 - Part 2: Advanced, https://bitbucket.org/openid/fapi/src/master/Financial_API_WD_002.md.

A FAPI 2.0 Attacker Model

The following is an excerpt from the current draft of the FAPI 2.0 attacker model [Fe21b].

FAPI 2.0 profiles aim to ensure the security goals listed above for arbitrary combinations of the following attackers, potentially collaborating to reach a common goal:[6]

Basic Attackers

A1 - Web Attacker: Standard web attacker model. Can send and receive messages just like any other party controlling one or more endpoints on the internet. Can participate in protocols flows as a normal user. Can use arbitrary tools (e.g., browser developer tools, custom software, local interception proxies) on their own endpoints to tamper with messages and assemble new messages. Can send links to honest users that are then visited by these users. This means that the web attacker has the ability to cause, arbitrary requests from users' browsers, as long as the contents are known to the attacker.

Cannot intercept or block messages sent between other parties, and cannot break cryptography unless the attacker has learned the respective decryption keys. Deviating from the common web attacker model, A1 cannot play the role of a legitimate AS in the ecosystem (see A1a).

A1a - Web Attacker (participating as AS): Like the web attacker A1, but can also participate as an AS in the ecosystem. Note that this AS can reuse/replay messages it has received from honest ASs and can send users to endpoints of honest ASs.

[6] The enumeration is not linear, as some attackers have been merged into others or split up into more detailed models over time.

A2 - Network attacker: Controls the whole network (like a rogue WiFi access point or any other compromised network node). Can intercept, block, and tamper with messages intended for other people, but cannot break cryptography unless the attacker has learned the respective decryption keys.

Note: Most attacks that are exclusive to this kind of attacker can be defended against by using transport layer protection like TLS.

Attackers at the Authorization Endpoint

The attackers for the authorization request are more fine-grained than those for the token endpoint and resource endpoint, since these messages pass through the complex environment of the user's browser/app/OS with a larger attack surface. This demands for a more fine-grained analysis.

A3a - Read Authorization Request: The capabilities of the web attacker, but can also read the authorization request sent in the front channel from a user's browser to the authorization server. This might happen on mobile operating systems (where apps can register for URLs), on all operating systems through the browser history, or due to Cross-Site Scripting on the AS. There have been cases where anti-virus software intercepts TLS connections and stores/analyzes URLs.

A3b - Read Authorization Response: The capabilities of the web attacker, but can also read the authorization response. This can happen e.g., due to the URL leaking in proxy logs, web browser logs, web browser history, or on mobile operating systems.

Attackers at the Token Endpoint

A5 - Read and Tamper with Token Requests and Responses: This attacker makes the client use a token endpoint that is not the one of the honest AS. This attacker can read and tamper with messages sent to and from this token endpoint that the client thinks as of an honest AS.

Note: When the token endpoint address is obtained from an authoritative source and via a protected channel, e.g., through OAuth Metadata obtained from the honest AS, this attacker is not relevant.

Attackers at the Resource Server

A7 - Read Resource Requests and Responses: The capabilities of the web attacker, but this attacker can also read requests sent to and from the resource server, for example because the attacker can read TLS intercepting proxy logs on the RS's side.

A8 - Tamper with Resource Responses: The capabilities of A7, but this attacker can also tamper with responses from the resource servers (e.g., a compromised reverse proxy in front of the resource server).