Keycloak における OAuth 2.0 ベースのセキュリティプロファイルの柔軟な実現方法

乗松 隆志[1,*]　中村 雄一[1]

概要：OSS である Keycloak で OAuth 2.0 ベースのセキュリティプロファイルを実現するにあたり，本体の改造が必要に必要になり，また設定作業が困難な課題がある．本論文では，その課題を解決し，かつ複数のセキュリティプロファイルの適用と，将来導入されるであろう新たなセキュリティプロファイルの適用を容易に実施できる方法である Client Policy を提案する．また，Client Policy の実装及びその適用結果について報告する．

キーワード：OAuth 2.0，Keycloak，セキュリティプロファイル

Flexible Way for Realizing OAuth 2.0 Based Security Profiles on Keycloak

Takashi Norimatsu[1,*]　Yuichi Nakamura[1]

Abstract: We have tried to support OAuth 2.0 based security profiles to Keycloak and found some issues. In this paper, we propose a flexible method to resolve these issues and also support multiple security profiles and newly introduced ones that might be defined in the future. Finally, we report the result of this method's evaluation.

Keywords: OAuth 2.0, Keycloak, Security Profile

1. はじめに

OAuth 2.0 という Web ベースの認可プロトコル[1]がある．近年では，[2]に示されるように，REST API 連携でこの OAuth 2.0 が広く使われるようになっている．OAuth 2.0 では，エンドユーザー，エンドユーザーが保有するリソースへのアクセスを API で提供するリソースサーバー，API をコールするクライアント，エンドユーザーからの認可を取得し，トークンと呼ばれる認可情報を発行・管理する認可サーバーというエンティティが登場する．ここで要となるエンティティは認可サーバーである．認可サーバーに対して様々なエンドポイントが定義されており，クライアントはこれらエンドポイントをコールする中で，エンドユーザーによる認可の結果としてトークンがクライアントに発行される．

1.1 OAuth 2.0 とセキュリティプロファイル

OAuth 2.0 はこれらのエンティティ間で認可情報をやり取りするフレームワークであり，様々な使い方が可能である．使い方を誤るとセキュリティホールが作りこまれることもあるため，OAuth2.0 の具体的な使い方の規定としてのプロファイルが求められるようになった．特にセキュリティを重視したプロファイルはセキュリティプロファイルと呼ばれ[3]，具体的なセキュリティプロファイルも，標準化団体により複数定義されている[4][5][6]．例えば，金融セクタでの利用など高いセキュリティを要求されるセキュリティプロファイルの標準仕様としては，OpenID Foundation による FAPI Security Profile[4]や英国 Open Banking によるセキュリティプロファイル[5], オーストラリア Consumer Data Right によるセキュリティプロファイル[6]が挙げられる．他，ネイティブアプリや Single Page Application （SPA）で OAuth 2.0 を安全に使用するための標準仕様[7][8]もあり，これらも広義ではセキュリティプロファイルとみなすことができる．

これらのセキュリティプロファイルでは, OAuth 2.0 に登場するエンティティとエンドポイントに対して種々の要件を定めている．すなわち，認可サーバーが公開するエンドポイントに対する要件があり，そのエンドポイントをどのようにクライアントやリソースサーバーが使うかを定めている．他，認可サーバーが発行したトークンをクライアント・リソースサーバがどう取り扱うかという要件も含まれている．また，これらのセキュリティプロファイルは固定ではなく，脆弱性発見に伴う修正などでバージョンアップされる場合もある．

1 株式会社 日立製作所
Hitachi, Ltd.
* takashi.norimatsu.ws@hitachi.com

1.2 認可サーバーとしての OSS : Keycloak

セキュリティプロファイルに対応したシステムを構築する際，認可サーバーがその要件に従った機能を有することが前提となっているため，認可サーバーの実装・設定が最も重要である．Identity Provider （IdP）や Identity and Access Management （IAM）用のソフトウェアが，OAuth2.0 の認可サーバーとしての機能を実装している．このようなソフトウェアとして，プロプライエタリなものもあれば，オープンソース・ソフトウェア（OSS）であるものもある．後者の例として，Keycloak[9]が挙げられる．Keycloak は OSS であり無償で利用できることから，世界中で広く利用されており[10]，商用利用向けに有償のサポートサービスも提供されている[11][12].

筆者らは，この Keycloak に対して，セキュリティプロファイルの一つである FAPI Security Profile[4]の実装を行っている．その過程で，従来の Keycloak でセキュリティプロファイルを実現するには，多大な設定が必要となる等，運用・構築上様々な課題があることが分かった．そこで，筆者らは，これらの課題を解決するために，Client Policy というコンセプトを導入しこれを Keycloak に実装し，この Client Policy を利用して FAPI Security Profile を実装することとした．本論文ではこの Client Policy とその有効性の検証を報告する．

2. Keycloak によるセキュリティプロファイル実現の課題

認可サーバーの観点からすると，セキュリティプロファイルの適用単位はクライアントとなる．Keycloak でセキュリティプロファイルをクライアント単位に適用するためには，セキュリティプロファイルに示された要求事項を満たすようにクライアントに対する設定項目や振舞いを変更する必要がある．一般にクライアントは複数あり，またクライアント毎に求められるセキュリティプロファイルが異なることもある．従来の Keycloak でクライアントにセキュリティプロファイルを適用する際には，次のような大きく 3 つの課題がある．

2.1 新規のセキュリティプロファイルのサポートや改変

新たにセキュリティプロファイルをサポートする，あるいは既存のセキュリティプロファイルを改定しようとする場合，足りない機能があれば Keycloak に実装する必要がある．しかし，本体に対して変更を行うと，他の部分に影響を及ぼす可能性があるため，デグレードを防ぐために本体全体でのテストが必要となり，その分工数がかかる．さらには，商用サポートサービスを受ける際の前提条件として，本体の改造が禁止されている場合もある．

仮に本体を修正できたとしても，修正済みの Keycloak にアップグレードするためには，動作中の Keycloak を停止させ，修正済みの Keycloak に置き換えて，再起動する必要がある．再起動が発生した場合，Keycloak が管理している Single Sign On （SSO）用のログインセッションが失われる．その為，Keycloak を IdP として使用し SSO を行っているユーザーは，再度ログインを行わなければならなくなり，ユーザビリティが損なわれる問題もある．

2.2 セキュリティプロファイルの多数のクライアントへの適用

セキュリティプロファイルを多数のクライアントに新規に適用・変更・削除する場合，セキュリティプロファイルを実現するための設定を，多数のクライアントに対して実施する必要がある．そのため，作業量が多く Keycloak の管理者が操作ミスを起こしやすい．

本論文執筆時の最新の keycloak である keycloak 11.0.0 を例とすると，レルムの設定の内，セキュリティプロファイルに関する項目は 48 項目あり，クライアントの設定の内，セキュリティプロファイルに関する項目は 43 項目ある．後者の 43 項目について，これらを登録時のみでなく，セキュリティプロファイルに関する項目の変更が生じた場合にすべてのクライアントに対して実施する必要がある．

2.3 複数のセキュリティプロファイルの実現

クライアントがコールする API によって，異なるセキュリティレベルを要求される場合がある．例えば，銀行口座の情報を読み取れる API と銀行口座から送金を行う API があった場合，前者よりも後者の方が高いセキュリティレベルを要求される．この場合，クライアントに対して複数のセキュリティプロファイルを適用し，コールする API によって適用するセキュリティプロファイルを変更する必要がある．

Keycloak では，クライアントやユーザーといった管理対象はレルムという単位で管理されている．レルム内で管理されるクライアントに対して適用される設定は 1 パターンのみである．よって，適用されるセキュリティプロファイルは 1 つに限定される．その為，前述のように複数のセキュリティプロファイルをクライアントに適用する場合，セキュリティプロファイル毎にレルムを設け，全てのレルムに同じクライアントやユーザーといった管理対象を登録し管理なければならなくなる．例えば，銀行口座の情報を読み取れる API 用のセキュリティプロファイルをサポートしたレルムと，銀行口座から送金を行う API 用のセキュリティプロファイルをサポートしたレルムとを設け，双方のレルムに同じクライアントやユーザーといった管理対象を登録する作業が必要となる．更に，それらの変更や削除があった場合，双方のレルムに対して同じ変更・削除を行う必要がある．

このように，管理対象の新規登録・変更・削除を複数のレルムに対し同じ内容で実施する必要がある．その為，作業量が多く Keycloak の管理者が操作ミスを起こしやすい．

3. Client Policy の提案

本論文では，下記の基本方針に従い前述の課題を解決するために Client Policy というコンセプトを導入する．

3.1 基本方針：Client Policy の概念

まず，2 章で述べた課題を解決するための基本方針を述べる．本論文では，課題を解決するために Client Policy という概念を提案する．

Client Policy は，Client Policy Basics と Policy から成り立つ．2.1 節で述べた課題が生じる根本原因は，セキュリティプロファイルを構成する設定の処理を，Keycloak 本体のコードで実装していることにある．そこで，これらの処理を Keycloak 本体から切り離し，動的にロード・アンロードするための機構である「Client Policy Basic」を設ける．その上に「Policy」という形でセキュリティプロファイルに関する処理を実装し，動的にロード・アンロードできるようにする．ここで，本体改修については Client Policy Basics のみに留めるようにし，Client Policy Basics を Keycloak コミュニティにパッチ投稿し，本体にマージすることで，将来版では本体改修を不要にすることを狙う．

2.2 節及び 2.3 節で述べた課題が生じる根本原因は，クライアントとその設定が一体化していることにある．そこで，Client Policy Basics の上の Policy では，これを分離し，設定は異なる論理的な管理対象とし，クライアントと設定との間を関係づけられるようにすることで，根本原因を取り除くこととする．

3.2 Client Policy の設計

3.1 節で述べた基本方針に従った Client Policy の設計について述べる．Client Policy とは，クライアントに対する設定および処理用のテンプレートに類似したフレームワークである．個々のクライアントに対して設定及び処理をするのではなく，クライアントの集合を規定し，これに対して設定及び処理を行うものである．

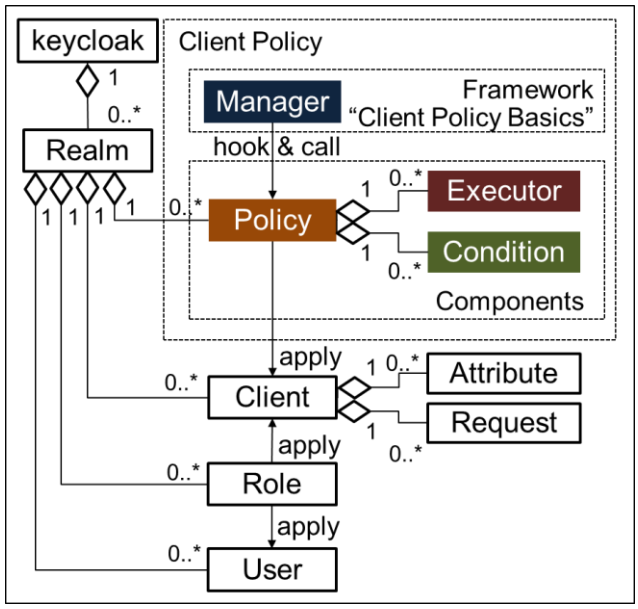

図 1　Client Policy のコンポーネント構成

図 1 に示されるように，Client Policy は，大きくは Client Policy Basics と Policy から成り立つ．Client Policy Basics は，その上で動作する Policy のロード・アンロードを管理するとともに Policy を実装するためのインタフェースであるフレームワークを提供する．

Policy にてセキュリティプロファイルをクライアントに適用する．Policy は Condition と Executor から構成される．Condition とは，あるクライアントがセキュリティプロファイルの適用対象であるかどうかを決めるコンポーネントである．Executor とは，Policy の適用内容を規定し，Condition で適用とされたクライアントに対してセキュリティプロファイルを実現するための処理を行う．

Policy は複数の Condition と複数の Executor を保持できる．この場合，全ての Condition で適用対象となったクライアントが，この Policy の適用対象となる．そして，このクライアントに対し，全ての Executor を実行する．

Condition により，セキュリティプロファイルの実現のための処理を実装した Executor を適用するクライアントの集合を定義できる．その為，クライアント一つ一つに対しセキュリティプロファイル実現のための設定を行う必要がない．

Keycloak のレルムは，複数の Policy を所持できる．よって，複数の Policy をクライアントに適用することができる．例えば，前述の FAPI-RW Security Profile は，同じく前述の FAPI-RO Security Profile の適用を前提として，追加で様々な要件を設けている．そこで，FAPI-RO Security Profile 用の Policy と，それに対する追加要件を示す FAPI-RW Security Profile 用の Policy 設け，この 2 つの Policy をクライアントに適用することで，本来の FAPI-RW Security Profile を適用することが考えられる．

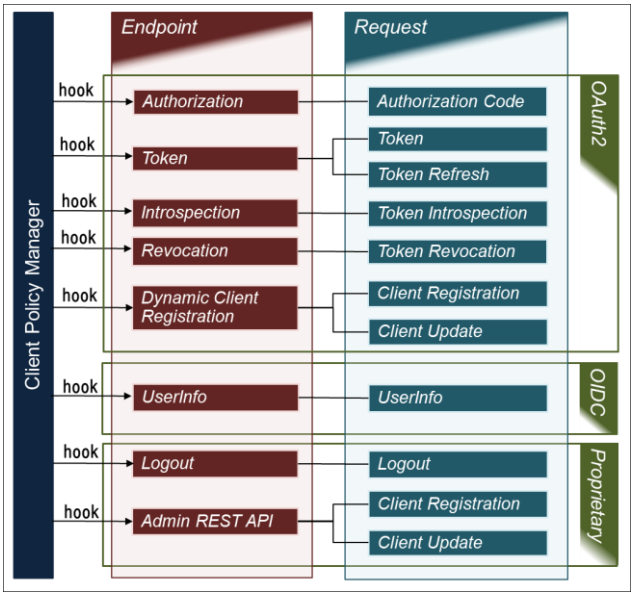

図 2　Client Policy の実行ポイント

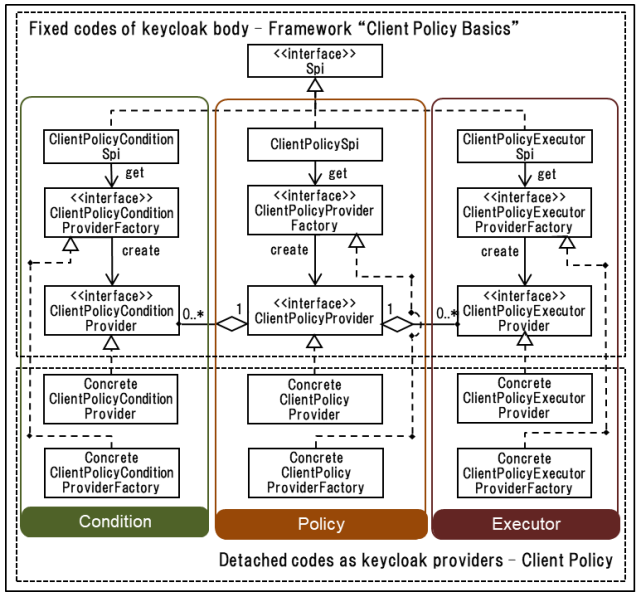

図 3　Client Policy のクラス構成

Policy は，クライアントからの要求を受け付ける箇所（図 2 の Endpoint）にて，クライアントの静的な特性（図 1 の Attribute）とクライアントからの動的な要求（図 1 及び図 2 の Request）を元にして Condition を評価し，Policy を適用するかどうかを決める．適用する場合，Executor を実行する．これにより，特定の条件を満たすクライアントとその要求に対してのみ，特定の処理を実施することが可能となる．よって，同じクライアントに対してその要求ごとに異なるセキュリティプロファイルを適用することが可能となる．

3.3 Client Policy の実装

実装面からみると，図 3 で示されるように，Policy や Condition や Executor は，Client Policy のフレームワークである Client Policy Basics が規定したインタフェースを実装したプラグイン形式のソフトウェアコンポーネントである．これを Java ではプロバイダと呼ぶ．これらプロバイダのビルドは Keycloak 自体のビルドと独立している．そして Keycloak の稼働中に動的にロード・アンロードが可能である．

Client Policy のフレームワークである Client Policy Basics のコードと，具体的なセキュリティプロファイルの実現のための Condition と Executor のコードは分離されている．その為，セキュリティプロファイルのバージョンアップや，新しいセキュリティプロファイルをサポートする場合，プロバイダとしてこれらを実装すればよいので，Client Policy Basics さえ Keycloak 本体に入っていれば，Keycloak 本体のコードを修正する必要がない．

セキュリティプロファイルのバージョンアップや，新しいセキュリティプロファイルをサポートする場合，これに対応する Condition や Executor を実装して Keycloak で稼働させる必要がある．これらはプロバイダとして実装されるため，Keycloak の稼働中にロードすることが可能である．よって，Keycloak を停止させることなくセキュリティプロファイルのバージョンアップや，新しいセキュリティプロファイルに対応できる．

4. 検証結果

2 章で述べた課題が，Client Policy によって解決したかどうかを検証した．その結果を報告する．

4.1 複数のクライアントに複数のセキュリティプロファイルを適用する場合の Keycloak 管理者の作業項目数

2.2 節及び 2.3 節で述べた課題が，Client Policy により解決したかどうかを検証するために，従来の Keycloak と Client Policy 実装後の Keycloak とで，複数のクライアントに複数のセキュリティプロファイルを適用する場合の Keycloak 管理者の作業項目数を比較する．

- Client Policy 適用以前

各レルムにおいて，クライアントの作成・設定，ユーザーの作成・設定，適用対象のセキュリティプロファイルの

設定の作業があるため，作業項目数は以下となる．

適用対象のセキュリティプロファイル数
x （レルムの作成・設定
+ ユーザー数 x ユーザーの作成・設定
+ クライアント数
x （クライアントの作成・設定
+ 適用対象のセキュリティプロファイル設定））

作業項目は，適用対象のセキュリティプロファイル数 x（クライアント数 + ユーザー数）に概ね比例して増加する．

- Client Policy 適用後

レルムの作成・設定
+ クライアント数 x クライアントの作成・設定
+ ユーザー数 x ユーザーの作成・設定
+ 適用対象のセキュリティプロファイル数
x 適用対象のセキュリティプロファイルの設定

作業項目は，（適用対象のセキュリティプロファイル数 + クライアント数 + ユーザー数） に概ね比例して増加する．

Client Policy 適用前と適用後の作業項目数と比較すると，適用後の方が作業項目数を低減できていることが分かる．

4.2 新規のプロファイルの実現

2.1 節で述べた課題が，Client Policy により解決したかどうかを検証する．

まず，本体の改造については，Client Policy のフレームワークである Client Policy Basics について，Keycloak コミュニティにパッチ投稿し，Keycloak 本体のコードにマージされた[13]．よって以降のバージョンにおいては本体の改造は不要となる．

次に，動的なセキュリティプロファイルの適用について，Client Policy Basic のフレームワークの上に，新規セキュリティプロファイルを実現するための Policy（Condition, と Executor）を実装し，これらを Keycloak 稼働中にロードし，セキュリティプロファイルをクライアントに適用できるかを評価する．

4.2.1 評価対象プロファイル

評価の対象とするプロファイルは FAPI-RO セキュリティプロファイルの一部である RFC 7636 Proof Key for Code Exchange by OAuth Public Clients（PKCE）[14]への対応とする．すなわち，クライアントは認可エンドポイントへのアクセス時に PKCE を利用することを必須とするものである．

4.2.2 必要な実装

本プロファイルの実現のための Policy として，Condition と Executor を次のように実装した．

- Condition

Policy の対象となるクライアントを定義するため，「TestClientRolesCondition」という予め指定されたロールを付与されたクライアントを Policy の適用対象にするプロバイダを実装した

- Executor

FAPI-RO セキュリティプロファイルの要求事項のうちの 1 つを実現する「TestPKCEEnforceExecutor」をプロバイダ形式で実装した．すなわち，Policy 適用対象のクライアントが PKCE プロトコルに沿って通信しているかをチェックし，沿っていない場合通信をエラーとするプロバイダである．

- Policy

フレームワークである Client Policy Basic が提供するデフォルトのクラスを利用した．

4.2.3 評価方法

Keycloak が内包している統合テストフレームワークである Arquillian Integration Test を利用し，上記の Condition, Executor，Policy を Keycloak 稼働中にロードし，これをクライアントに対して適用するテストコードを実行することで検証とすることとした．

以下のテストシナリオを Arquillian Integration Test で実装し実行する．

1. PKCE プロトコルに従わない Authorization Code Grant の実施。
   期待値：今だ Client Policy が存在しないため，成功する．
2. TestClientRolesCondition 及び TestPKCEEnforceExecutor を保持する Client Policy のロード。
   期待値：PKCE プロトコルに沿って通信しているかどうかをチェックする Client Policy が動的にロードされる．
3. PKCE プロトコルに従わない Authorization Code Grant の実施。
   期待値：Client Policy が PKCE プロトコルのチェックを行うようになったため，失敗する．

4.2.4 結果

4.2.3 節で述べたテストシナリオを, ClientPolicyBasicsTest という Arquillian Integration Test 用のクラスの testCreateDeletePolicyRuntime メソッドに実装した．

このテストを実行したところ，4.2.3 節にある期待値通り，Client Policy をロード後クライアントにこれが適用され，PKCE のプロトコルに違反しているという旨でエラーとなった．

5. おわりに

Keycloak に OAuth 2.0 ベースのセキュリティプロファイルを適用するためには，コード本体の修正が必要であったり，設定が煩雑という課題がある．これら課題を解決するために，Client Policy というコンセプトを導入，そのフレームワーク部である Client Policy Basics を実装し，Keycloak の master ブランチに merge され，本体の修正が不要となり，設定も簡易化されていることを確認した．

謝辞　Keycloak の Project Lead である Stian Thorgersen 氏からは，Client Policy のコンセプト，その要件や設計議論で数多くのアイデアを頂きました[15][16]．ここに感謝します．

参考文献

[1] “RFC 6749 The OAuth 2.0.0 Authorization Framework”. https://tools.ietf.org/html/rfc6749, (参照 2020-07-03).
[2] セキュアなシステム間連携を支える OSS 認証認可技術．日立評論 2020 vol.102 No.3. https://www.hitachihyoron.com/jp/archive/2020s/2020/03/03a04/index.html
[3] “An Extensive Formal Security Analysis of the OpenID Financial-Grade API”. https://ieeexplore.ieee.org/document/8835218, (参照 2020-07-03).
[4] “Financial-grade API - Part 1: Read-Only API Security Profile“. https://openid.net/specs/openid-financial-api-part-1-ID2.html, (参照 2020-07-03).
[5] “OpenBanking Security Profiles”. https://standards.openbanking.org.uk/security-profiles/, (参照 2020-07-03).
[6] “Consumer Data Right Security Profile”. https://consumerdatastandardsaustralia.github.io/standards/#security-profile, (参照 2020-07-03).
[7] “RFC 8252 OAuth 2.0.0 for Native App“. https://tools.ietf.org/html/rfc8252, (参照 2020-07-03).
[8] “OAuth 2.0.0 for Browser-Based Apps” . https://tools.ietf.org/html/draft-ietf-oauth-browser-based-apps-05, (参照 2020-07-03).
[9] “Keycloak”. https://www.keycloak.org/, (参照 2020-07-03).
[10] “Keycloak proposal for submission to CNCF”. https://github.com/cncf/toc/pull/405, (参照 2020-07-03).
[11] “Red Hat Single Sign On”. https://access.redhat.com/products/red-hat-single-sign-on, (参照 2020-07-03).
[12] “OpenStandia”. https://www.nri.com/jp/news/info/cc/lst/2018/0119, (参照 2020-07-03).
[13] “KEYCLOAK-14189 Client Policy : Basics” . https://github.com/keycloak/keycloak/pull/7104, (参照 2020-08-01).
[14] “RFC 7636 Proof Key for Code Exchange by OAuth Public Clients “. https://tools.ietf.org/html/rfc7636, (参照 2020-07-03).
[15] “KEYCLOAK-11612 Client conformance profiles”. https://github.com/keycloak/keycloak-community/pull/44, (参照 2020-07-28).
[16] “KEYCLOAK-13933 Client Policies”. https://github.com/keycloak/keycloak-community/pull/99, (参照 2020-08-04).

付録. テストシナリオの実行方法

4.2.3 節で述べたテストシナリオを実行する．下記コマンドを実行し Keycloak をビルドする．

```
> git clone https://github.com/keycloak/keycloak.git
> cd keycloak
> git checkout e0fbfa722ec70c429c3e8da19e5298dfbe9c57c1
> mvn install -Pdistribution -DskipTests
```

integration-arquillian/tests/base/src/test/resources/log4j.properties の下記の箇所のコメントアウトを解除しログ出力を有効とする．

```
log4j.logger.org.keycloak.services.clientpolicy=trace
log4j.logger.org.keycloak.testsuite.clientpolicy=trace
```

テスト用のコマンドを実行する．

```
> mvn -f testsuite/integration-arquillian/pom.xml ¥
  install -DskipTests
> mvn -f testsuite/integration-arquillian/tests/base/pom.xml ¥
  test -Dkeycloak.logging.level=info ¥
  -Dtest=org.keycloak.testsuite.client.ClientPolicyBasicsTest
  #testCreateDeletePolicyRuntime
```

上記コマンドを実行すると，下記のログ出力がなされる．

```
#triggerOnEvent,
 Client Policy Operation :
 event = AUTHORIZATION_REQUEST
#getProviders, Loaded Policy Name = MyPolicy
#doPolicyOperation, Policy Operation :
 name = MyPolicy, provider id = client-policy-provider
#getConditions, Loaded Condition
 id = 8657b817-9534-4a29-8e3f-6bbff9b3e92f,
 name = TestClientRolesCondition,
 provider id = test-clientroles-condition
#isSatisfied, POSITIVE :: This policy is applied.
#getExecutors, Loaded Executor
 id = b9b8164d-c835-482b-967d-23d2d1102bd1,
 name = TestPKCEEnforceExecutor,
 provider id = test-pkce-enforce-executor
#execute, EXECUTOR EXCEPTION :
 name = TestPKCEEnforceExecutor,
 provider id = test-pkce-enforce-executor,
 error = invalid_request,
 error_detail = Missing parameter: code_challenge_method
```

これは，クライアントが認可エンドポイントに認可要求を出したが PKCE プロトコルに従っておらず，エラー応答となったことを示している．