Auth0

Verified Email Address Change

Customized Extensibility Design

Auth0

Contents

Document Control

Revision

Author	Date	Version	Comment
Peter Fernandez	29th August 2020	*2.4.0*	*Update to problem description & actual address change workflow*
Peter Fernandez	10th August 2020	*2.3.0*	*Update Email Address Change initiation & associated diagrams*
Peter Fernandez	20th June 2020	*2.2.0*	*Refactor re-authentication workflow*
Peter Fernandez	3rd June 2020	*2.1.2*	*Refactor of Executive Summary for additional clarity.*
Peter Fernandez	1st June 2020	*2.1.1*	*Refactor Executive Summary for improved clarity.*
Peter Fernandez	26th May 2020	*2.1.0*	*Refactor for redirect section 7.*
Peter Fernandez	26th May 2020	*2.0.2*	*Verification section update.*
Peter Fernandez	26th May 2020	*2.0.1*	*Update to API call guidance in 6.*
Peter Fernandez	14th May 2020	*2.0.0*	*Release version*

Executive Summary

Self-service typically provides users with the ability to change one or more aspects of their user profile. Commonly referred to as *MyAccount* or *MyProfile* functionality, this often includes the capability for a user to change his/her email address in cases where an existing email address is no longer valid or preferred. Self-service email address change without the proper controls however can lead to various account management issues, and/or expose security vulnerabilities which can be exploited.

Problem

Out of the box, the Auth0 Management API provides email address change for a user identity stored in a Database (or Custom Database) Connection. This API however performs an ***immediate*** change of email address: though some basic email address validations are provided, there is no provision for verification of an email address before the change is made.

Use of the Auth0 Management API then without proper security and/or validation checks can be problematic. For example, an unnoticed typo in an email address could mean that a user ends up locking themselves out of their account - especially in cases where incorrect entry goes undetected before a user ends their current login session. Worse still, an incorrectly entered email address may mean that a user unintentionally provides account access to another person - who might then use, say, password reset to gain control of that account. Whilst the use of Multi-Factor Authentication (MFA) can prevent others from “stealing” an account, it’s a heavy-weight solution and does nothing to solve the issue of user account self-lockout.

Solution

The design described in this document is provided by Auth0 Professional Services as a Custom Implementation, which leverages Auth0 functionality to address the problem by essentially requiring a user to **re-authenticate** in a new email context *before* the actual email address change is performed. This strategy can also provide protection in the case where the new email address is claimed *before* workflow is complete.

Whilst the design described is primarily for use within a user self service context, the workflow described here-in could be adapted for use in other contexts where change of an the email address is required.

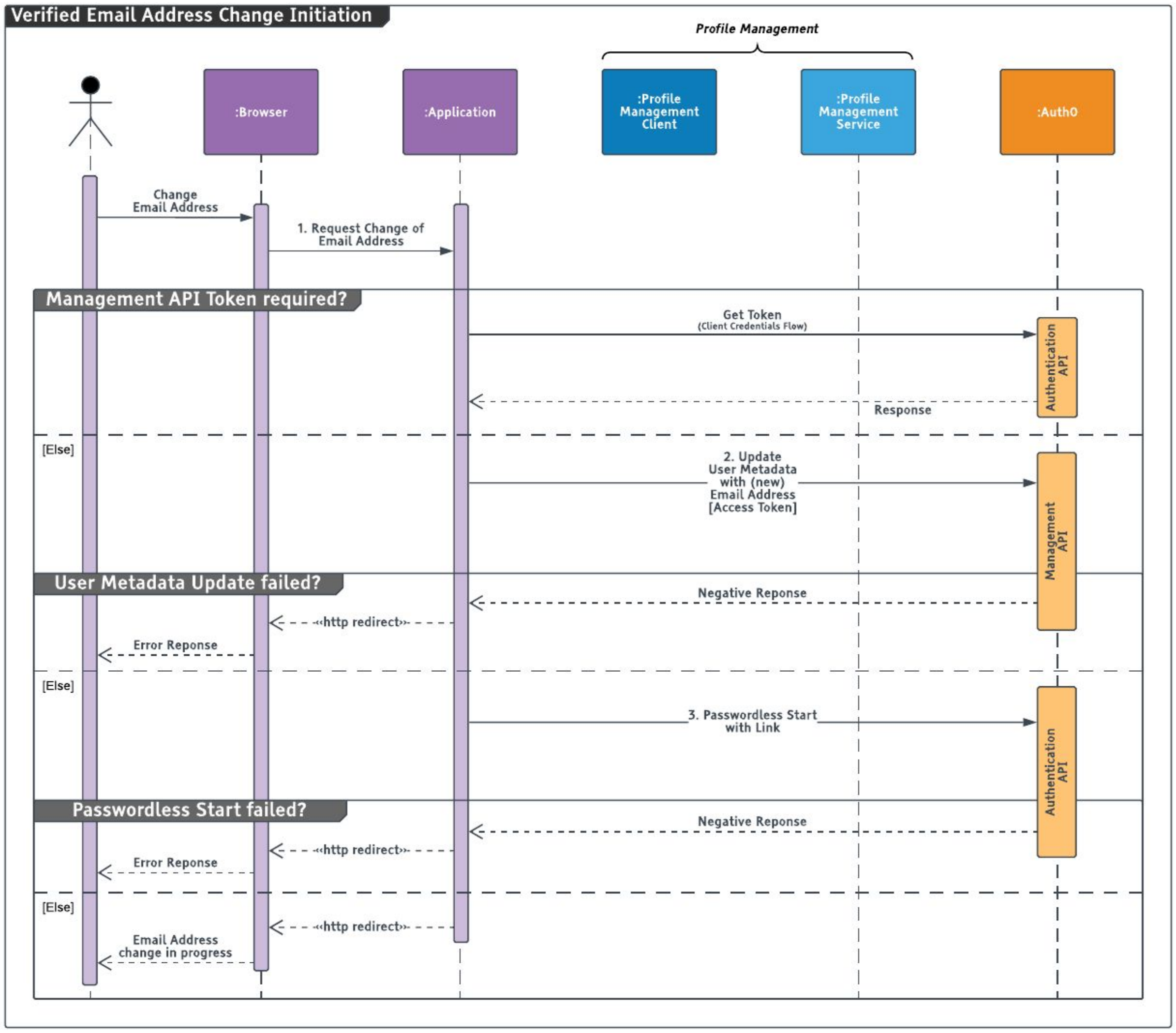
The design as described can be implemented by your own development team. Alternatively the Professional Services team at Auth0 can be engaged to provide Custom Implementation Services (CIS) in order to create a turn-key solution where this, in conjunction with other workflows, could be implemented.

Design

Workflow for Verified Email Address Change is essentially split into two discrete stages, each of which makes use of the features available in Auth0 in order to achieve the desired outcome.

Email Address Change initiation

As can be seen from the diagram below, the first stage of the process makes use of some *MyAccount* (a.k.a. *MyProfile*) implementation to provide functionality to initiate the email address change. The design of such functionality is beyond the scope of this document, suffice to say it would typically be implemented in either a web based Single Page Application, as a page in a Regular Web application, or as part of some Mobile/Native application.


Verified Email Address Change Initiation
Profile Management
:Browser
:Application
:Profile Management Client
:Profile Management Service
:Auth0
Change Email Address
1. Request Change of Email Address
Management API Token required?
Get Token (Client Credentials Flow)
Authentication API
Response
[Else]
2. Update User Metadata with (new) Email Address [Access Token]
Management API
User Metadata Update failed?
Negative Reponse
«http redirect»
Error Reponse
[Else]
3. Passwordless Start with Link
Authentication API
Passwordless Start failed?
Negative Reponse
«http redirect»
Error Reponse
[Else]
«http redirect»
Email Address change in progress

However *MyAccount*/*MyProfile* functionality is implemented, some method allowing the user to change his/her email address needs to be provided that would initiate email address change in the following manner:

1. A secure request would be made to some proprietary email address change endpoint. This could be via a call to a backend - in the case of a Regular Web Application - or, preferably, via some independent *Profile Management* application - either already in existence or purpose built.

> *Use of an independant Profile Management Client, together with a corresponding Profile Management Service - e.g. Backend for Frontend (BFF), or more traditional web service - allows common Profile Management functionality to be built that could be easily and securely shared across multiple applications.*

2. Email address change request handling would initiate the update of `user_metadata` with the new email address desired. In the diagram above, this is performed from the backend of a Regular Web Application.

   2.1. Metadata would be updated for the existing user who initiated the email address change, and would be performed via use of the Auth0 Management API. With the intent being to *initiate* email address change in the first stage of workflow, it makes sense to use metadata associated with the already existing user as a good place to temporarily store the requested new email address

   2.2. Use of the Auth0 Management API would require a suitable Access Token. As in the above diagram, this can be obtained via a call, say, to the Auth0 Authentication API, using the `/token` endpoint with Client Credentials Flow.

      2.2.1. Auth0 Management API access token caching et al should be employed where possible, thus mitigating unnecessary calls to Auth0 APIs (Auth0 APIs being subject to Rate Limiting Policy) and/or token generation.

      2.2.2. Though not depicted in the above diagram, error conditions should also be catered for - including errors generated as a result of rate limiting.

   2.3. Storage of the new email address would typically involve creating, say, a `newEmail` definition in `user_metadata`, that would be subsequently read and processed later. The definition would contain the new email address for the user, to be used - as indicated - later in the process (see 7.1.2 below). Use of metadata also limits the amount of additional processing that's required; metadata for a user is incorporated as part of the User Profile (a.k.a. User Account) information that follows a user through the flow of processing in the Auth0 Engine.

Use of `user_metadata` typically requires no elevated privileges beyond what a user would typically be able to attain: `user_metadata` is explicitly designed for user read/write access with little more than an authenticated set of credentials. For application enforced data, use of `app_metadata` should be preferred; in this case, whilst there is an application which provides a UI for the user to change his/her email address, the new address a user chooses is not application enforced.

3. Upon successful update of user metadata, email address change request handling would then use the Auth0 Authentication API to initiate a sign-in using a Passwordless Connection. This would be email type passwordless sign-in, using the *new* email address requested and utilizing email Magic Link functionality.

   3.1. Passwordless sign-in should be initiated using the `client_id` associated with some seperate *Profile Management* application definition in Auth0. This is a separate Application definition in Auth0, irrespective of whether an actual existing application is used or not. The definition limits the scope for which Passwordless connections are enabled: arbitrarily enabling passwordless for an application is not desirable, so having a unique application context works well.

      3.1.1. Using a separate *Profile Management* application definition in Auth0 also provides a mechanism for quarantining invalid users in the case where a valid but incorrect new email address was entered.

   3.2. The call to the Auth0 Authentication API should include the `response_type` in the additional `authParams`.

      3.2.1. For email address change verification implemented in a Single Page Application (SPA) say, a `response_type` of `code` would typically be used, in conjunction with Authorization Code Flow with PKCE .

      3.2.2. For Regular Web Application implementation, a `response_type` of `code` would also be used. This would then be used with standard Authorization Code Flow.

Passwordless authentication with email magic link is inherently less secure, as no password credential is required as part of user authentication. In the Verified Email Address Change design presented, steps have been taken to employ Auth0 features, functionality and best practice, in an attempt to reduce security risk wherever possible.

In addition, use of a separate *Profile Management* application definition in Auth0 provides a way to identify the purpose for which a passwordless email was generated. By using the `application.clientID` in conjunction with the user metadata defined in 2 (above), it would be relatively easy to identify the purpose of the passwordless operation.

A separate Profile Management application definition in Auth0 also provides for secure processing context in accordance with the principles of least privilege. A separate application definition not only defines context for execution, but also provides a clearly defined scope of context for isolation and/or quarantine should the need arise. A separate Profile Management Service application definition should also be defined.

Existence of user metadata created in 2 could also be used by *MyAccount/MyProfile* functionality to inform the user of email address change progress - particularly if he/she navigates away from the *MyAccount/MyProfile* page before change occurs. Functionality could also be implemented to provide for `cancel` or even `update` operations, but is beyond the scope of this document.

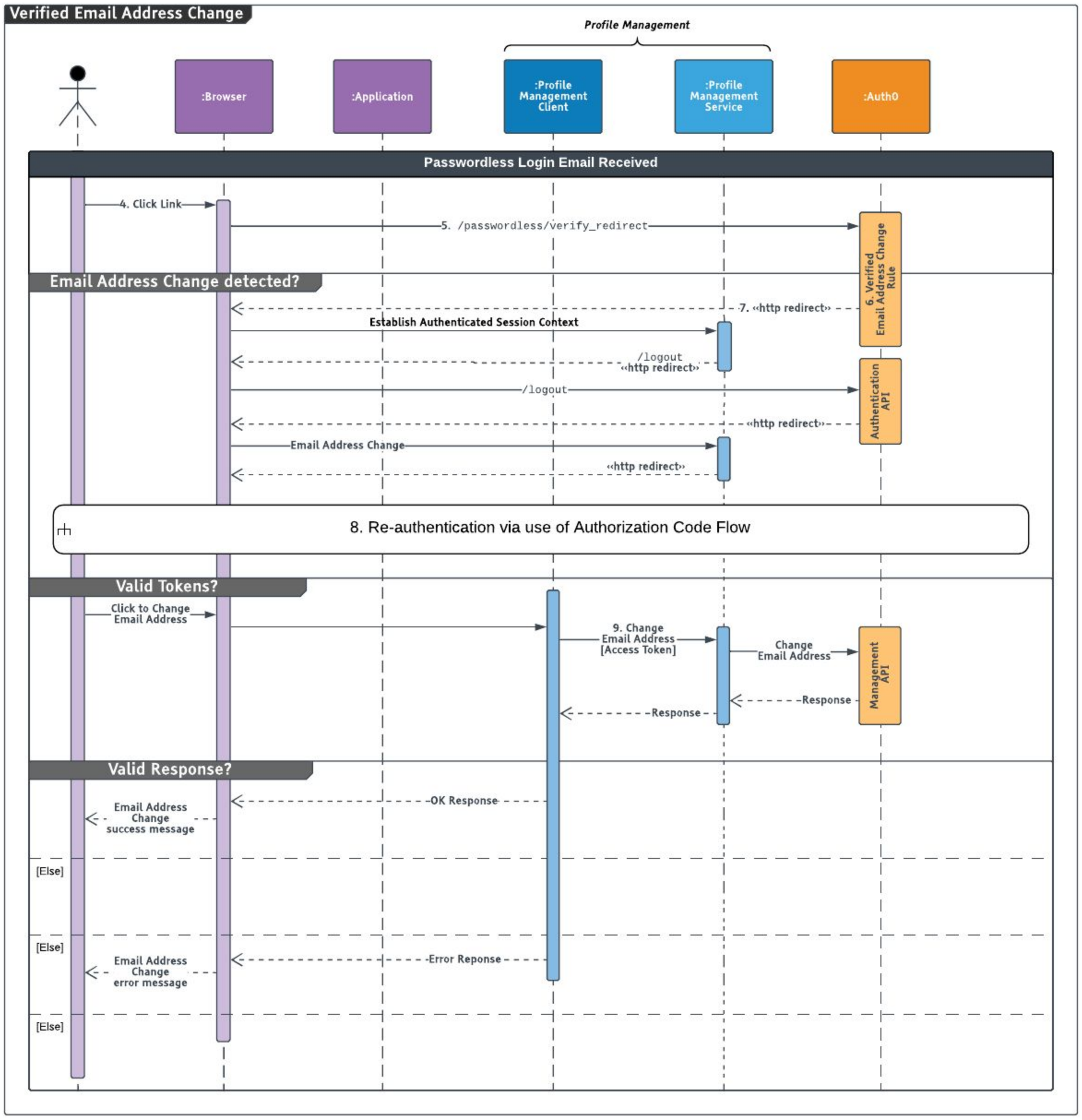
As discussed, email address initiation change could be implemented as part of existing MyArrount/MyProfile functionality. However, having some independent application would allow common Profile Management functionality to be built that could be easily and securely shared across multiple applications. The Profile Management Client and the Profile Management Service would typically be considered as complementary counterparts.

Assuming 2 & 3 above are successful, the user would typically be notified via UI that email address change is in progress. If a failure occurs at any point then the UI would ideally notify the user (of the failure) and the process of email address change would be terminated.

Both a Profile Management Service and a Profile Management Client would be protected by the use of id token and access token issued by Auth0 (see below for further details). As a best practice it's recommended that use of any existing MyAccount/MyProfile functionality is protected in a similar manner.

Passwordless sign-in initiated in 3 (above) will cause an email to be sent to the new email address for the user. This email will contain a magic link which when clicked will perform user login. The magic link can be opened in any browser instance, even in a browser associated with a device that's different to the one used to initiate email change. The format and content of the email is governed by the passwordless email template, which can be tailored using the common variables available to all email templates in Auth0.

Email Address Change


Verified Email Address Change
Profile Management
:Browser
:Application
:Profile Management Client
:Profile Management Service
:Auth0
Passwordless Login Email Received
4. Click Link
5. /passwordless/verify_redirect
6. Verified Email Address Change Rule
Email Address Change detected?
7. «http redirect»
Establish Authenticated Session Context
/logout «http redirect»
Authentication API
/logout
«http redirect»
Email Address Change
«http redirect»
8. Re-authentication via use of Authorization Code Flow
Valid Tokens?
Click to Change Email Address
9. Change Email Address [Access Token]
Change Email Address
Management API
Response
Response
Valid Response?
OK Response
Email Address Change success message
[Else]
[Else]
Email Address Change error message
Error Reponse
[Else]

The second stage of Verified Email Address Change follows on from the Email Address Change initiation stage (described above), and is where verification - as well as the actual change of email address - is performed. The flow in this part of the process is illustrated in the preceding diagram.

4. The user receives an email generated as a result of the operation described in 3 (above). The email contains a magic link, and instructions telling the user to click on the link in order to initiate the sign-in process; the magic link effectively acts as an event notification mechanism.

4.1. The email will typically follow the standard out-of-box Auth0 passwordless template, which can be further customized as described in the Auth0 documentation.

5. When the user clicks on the link in the email a passwordless sign-in is initiated.

5.1. Only one active user session per user agent (browser instance), per domain, is currently supported for an Auth0 tenant. Ordinarily, this would mean that a passwordless authentication transaction will change any existing authenticated user context - i.e. from a non-passwordless user context to a passwordless one. In the Verified Email Address Change design proposed however, passwordless authenticated user context is mitigated in a number of ways in order to reduce security risk, whilst still maintaining functionality:

5.1.1. by support for opening of the magic link in some other browser instance,

5.1.2. by the user being required to re-authenticate as part of the email address change verification process - which will ultimately set the user session back to a non-passwordless authenticated context.

5.1.3. by utilizing Auth0 Logout capability, reducing security risk by terminating any authenticated user context, passwordless or otherwise.

5.2. An alternative option would be to make use of separate sessions in different domains. In this case, the passwordless authentication transaction could occur in the base Auth0 tenant domain, while all other authentication transactions occur in some custom domain. This, of course, would require the use of Custom Domains in Auth0 (only currently available with paid subscription plans).

BEST PRACTICE *Whilst use of custom domains provides support for an alternative authentication context in the browser, the added use of Auth0 Logout provides a mechanism for reducing security risk should a user simply navigate away without completing Verified Email Address Change. In such a case, the use of Logout will mean that no authenticated user context, passwordless or otherwise, will persist in the browser.*

6. Sign-in will drive the Verified Email Address Change Rule. This is a rule declared in Auth0, with the explicit job of determining if a passwordless login was driven for the intention of a verified email change operation. The specific name of the rule is unimportant.

6.1. The rule detects if an email address change is pending by: (a) looking at `context.clientID` to determine if the `clientID` for which the login was driven is that of the *Profile Management* application definition in Auth0, and (b) looking to see if there is a corresponding user with the `newEmail` claim present in `user_metadata` (as described in 2 above).

6.1.1. Performing (b) would typically be achieved via the use of the Auth0 Management API user search functionality, in order to locate a user profile with corresponding metadata as created in 2. Any call to the Auth0 Management API will incur latency, and as the Management API is also rate limited, a call should only be made when absolutely necessary. Hence the `clientID` check in (a) is performed first; the Management API call will then only be executed if the calling context is associated with the *Profile Management* application definition in Auth0.

6.1.2. User search via use of the Auth0 Management API also provides improved UX support for situations where: (i) an email address change has been canceled prior to the rule being run, or (ii) where a further change of email address has been actioned, perhaps invalidating the change to the email address associated with the current passwordless user login.

Having a separate Profile Management application definition in Auth0 mitigates unnecessary API calls, by providing an easy-to-use identifier for context of execution. The number of API calls can be further reduced if Verified Email Address Change is the only functionality supported by passwordless event notification.

Redirection

7. If the rule in 6 (above) detects an email address change, then a redirect will be initiated. If no email address change is detected then execution would continue as normal.

   7.1. Redirect would be initiated by setting `context.redirect` as part of rule execution, and the redirection would be to some custom build UI. The exact details of this UI are beyond the scope of this document, however its corresponding URL should be secured via use of `HTTPS`.

Ideally, redirect would ultimately be via the Auth0 `/logout` endpoint. Redirect via the Auth0 `/logout` endpoint helps mitigate any lingering passwordless authentication context - particularly in cases where email address change is not completed for whatever reason (e.g. the user navigates away). As previously mentioned, Passwordless authentication with email magic link is inherently less secure, as no password credential is required as part of user authentication. So the use of Auth0 Logout mitigates against a number of potentially security vulnerabilities - particularly in cases where email address change is not completed for whatever reason.

Indirect redirection via the `/logout` endpoint is also preferred. Indirect `/logout` redirection would ideally occur via an endpoint in the *Profile Management Service*. Via the use of SSO, this would give the *Profile Management Service* the opportunity to obtain and process any additional information, as well as establish a secure and verifiable browser session context in which the email address change would occur.

An indirect call to the `/logout` endpoint opens up a number of possibilities, including the establishment of a verifiable browser session context in which the actual email address change would occur. The use of something like passport and the Auth0 passport authentication strategy would allow a verifiable browser session context to be established by leveraging SSO as part of the re-direct.

The `returnTo` parameter associated with a call to the Auth0 `/logout` endpoint would also be set, typically to the URL of the appropriate email address change endpoint in the *Profile Management Service* or *Profile Management Client* (depending on implementation and/or any additional processing required). This URL would typically also be included in the list of Allowed Logout URLs defined to the *Profile Management Service* definition in Auth0.

In addition to a Profile Management application definition in Auth0, the use of a separate Profile Management Service application definition provides for an additional context point for execution, both within Auth0 and within Profile Management processing.

Re-authentication

8. Upon successful completion of the rule pipeline, and if email address change was detected, then redirection would occur (as described in 7 above). Detecting no authenticated user context (due to redirect via `/logout`), the *Profile Management* application would typically redirect to Universal Login to request authentication. The client context for this would typically be that of the *Profile Management* application definition in Auth0; if the *Profile Management Client* is implemented as an independent SPA then ideally Authorization Code Flow with PKCE should be utilized, if a regular web application then Authorization Code Flow would typically be used (see Appendix A for further details).

8.1. The application could additionally present some message to the user before initiating authentication, however as contextual information will typically not be available, any message would likely be rudimentary. Any such messaging would be better placed in the email generated in 3 (above), and as previously mentioned this could be implemented using email template customization.

8.2. Any `connection` parameter passed as part of redirection (7; above) would also typically be used when redirecting to Universal Login, and would indicate the connection to be used as part of the authentication process.

Email address change can only be performed for a Database/Custom Database Connection in Auth0, so authentication ideally needs to be for the identity associated with that connection. An alternative to passing a `connection` parameter would be to have multiple Profile Management application endpoints - one for each possible Database/Custom Database connection. However where many Auth0 connections are defined, such a mechanism would become unwieldy.

8.3. Successful authentication would again trigger execution of the Verified Email Address Change Rule. Again, the rule would examine (a) `context.clientID` to determine if the clientID for which the login was driven was that of the *Profile Management* application definition in Auth0, and (b) look to see if there is a corresponding email claim present in `user_metadata` (as described in 2 above).

8.3.1. Assuming both (a) and (b) are true, the rule would then search for a passwordless user, with an email address equivalent to that of the *new email address* contained in the metadata. This would again typically be achieved via the use of the Auth0 Management API user search functionality, in order to locate such a user profile. As discussed previously, any call to the Auth0 Management API will incur latency, and as the Management API is also rate limited, a call should only be made when absolutely necessary.

8.4. If no corresponding match from 8.3.1. is found, then execution would continue as normal. If a match is discovered, then the rule would typically add custom claim(s) to both the `accessToken` and the `idToken` being generated - each one accessible via the context object.

8.4.1. The name(s) of the claim(s) used is unimportant so long as it/they are used consistently. The claim(s) would typically describe one or more of the following:

8.4.1.1. the name of the Database/Custom Database connection associated with the email address that is changing,

8.4.1.2. the `user_id` of the corresponding passwordless user that initiated the change, and

8.4.1.3. the new email address requested.

Replacing the email address

9. Acquiring tokens, *Profile Management* would then decode the id token to detect if the custom claim(s) described in 8.4 (above) are present. If the custom claim is not present or not of the correct format then no further processing should take place: an appropriate message should be displayed to the user and execution should terminate.

9.1. If the custom claim is present, and in the correct format, then *Profile Management* would initiate the process of performing actual email address change. Typically this would be done via the use of the *Profile Management Service*, with the access token obtained in 8 (above) being supplied as `Authorization: Bearer` token.

9.1.1. This obviously will require an API definition in Auth0, ideally with an `update:email`, say `scope`. This could be a separate API definition or part of an existing definition, either way the associated API `audience` would be specified as part of the re-authentication in 8 (above).

9.2. The name of the *Profile Management Service* end-point is immaterial, however it's suggested that it's called via `patch` operation in order to signify update/change.

9.3. The *Profile Management Service* end-point implementation would validate the bearer token supplied, and use the information in it to verify that the email address change is legitimate. This would typically involve examination of custom claims, and the previous session context established during 7 (above) provides for an additional layer of security checking.

9.3.1. Once this has completed successfully, a `patch` to the Auth0 Management API `users` endpoint would be made to update/replace the email address for the user; this can also be used to delete the email address change metadata associated with the user in 2 (above).

9.3.2. As always, recommended best practice for any calls to the Auth0 Management API is that they're done via the use of an appropriate Auth0 SDK - such as the `auth0-js` library for Node.

9.4. Finally the passwordless user can be deleted via a `delete` on the Auth0 Management API `users` endpoint.

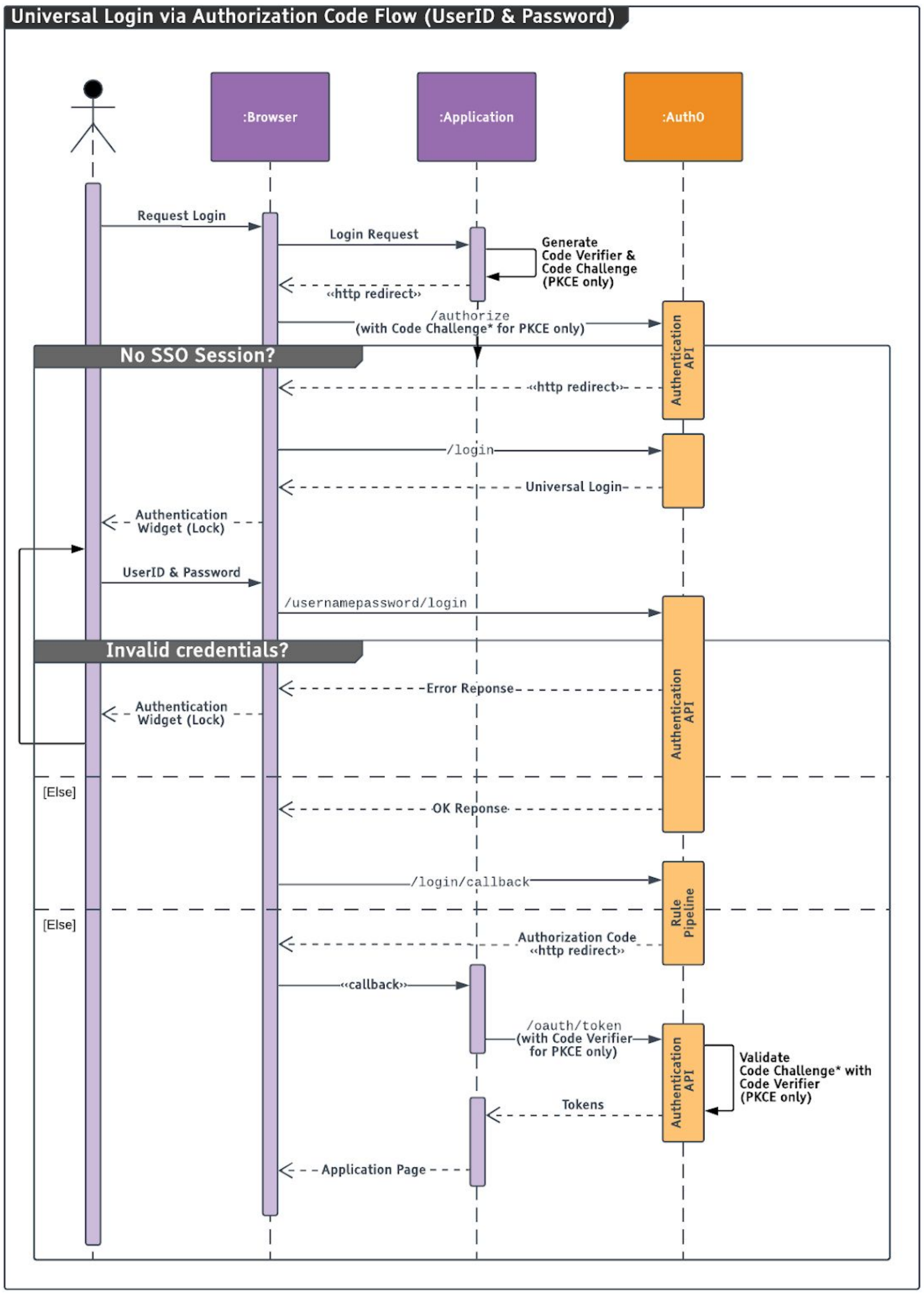
Use of verifiable session browser context information obtained via use of SSO (as described in 7, above), also provides the Profile Management Service with additional context information that can be utilized during the email address change verification process.

Assuming that 4 thru 9 execute successfully, the user would typically be notified that their email address was successfully changed. If a failure occurs at any point then the user should ideally be notified (of the failure) and the process of email address change would be terminated.

Ordinarily any Auth0 redirect from Rule would be completed with a redirection to the Auth0 `/continue` endpoint; all rules will be run again, with `context.protocol` typically being used to ignore specific processing (i.e. by comparing it to the value `redirect-callback`). However, in this case, no redirection to the `/continue` endpoint needs be performed, and the Rule pipeline should be left to be timed-out normally by Auth0.

Appendix A

Re-authentication via use of Authorization Code Flow


Universal Login via Authorization Code Flow (UserID & Password)
:Browser
:Application
:Auth0
Request Login
Login Request
Generate Code Verifier & Code Challenge (PKCE only)
«http redirect»
/authorize (with Code Challenge* for PKCE only)
No SSO Session?
Authentication API
«http redirect»
/login
Universal Login
Authentication Widget (Lock)
UserID & Password
/usernamepassword/login
Invalid credentials?
Error Reponse
Authentication Widget (Lock)
[Else]
OK Reponse
/login/callback
Rule Pipeline
[Else]
Authorization Code «http redirect»
«callback»
/oauth/token (with Code Verifier for PKCE only)
Validate Code Challenge* with Code Verifier (PKCE only)
Tokens
Application Page