LoginDiscoveryHandler Interface

Salesforce gives you the ability to log in users based on other verification methods than username and password. For example, it can prompt users to log in with their email, phone number, or another identifier like a Federation ID or device identifier. Login Discovery is available to these licenses: Customer Community, Customer Community Plus, External Identity, Partner Community, and Partner Community Plus.

Namespace

Auth

Usage

Implement a `Auth.LoginDiscoveryHandler` for an interview-based log in. The handler looks up a use from the identifier entered, and can call `Site.passwordlessLogin` to determine which credential to use, such as email or SMS. Or the handler can redirect a user to a third-party identity provider for login. With this handler, the login page doesn't show a password field. However, you can use `Site.passwordlessLogin` to then prompt for a password.

From the user perspective, the user enters an identifier at the log in prompt. Then the user completes the login by entering a PIN or password. Or, if SSO-enabled, the user bypasses login.

For an example, see LoginDiscoveryHandler Example Implementation. For more details, see Salesforce Customer Identity in *Salesforce Help*.

- **LoginDiscoveryHandler Method**
- **LoginDiscoveryHandler Example Implementation**

LoginDiscoveryHandler Method

Here's the method for `LoginDiscoveryHandler`.

- **login(identifier, startUrl, requestAttributes)**
  Log in the customer or partner given the specified identifier, such as email or phone number. If successful, redirect the user to the Experience Cloud site page specified by the start URL.

login(identifier, startUrl, requestAttributes)

Log in the customer or partner given the specified identifier, such as email or phone number. If successful, redirect the user to the Experience Cloud site page specified by the start URL.

Signature

```
public System.PageReference login(String identifier, String startUrl,
Map<String,String>requestAttributes)
```

Parameters

identifier

Type: String

Type: String

Path to the Experience Cloud site page requested by the customer or partner. The user is redirected to this location after successful login.

requestAttributes

Type: Map<String,String>

Information about the login request based on the user's browser state when accessing the login page. `requestAttributes` passes in the CommunityUrl, IpAddress, UserAgent, Platform, Application, City, Country, and Subdivision values. The City, Country, and Subdivision values com from IP geolocation.

Return Value

Type: System.PageReference

The URL of the page where the user is redirected.

Example

Here's a sample `requestAttributes` response.

```
CommunityUrl=http://my-developer-edition.mycompany.com:5555/discover
IpAddress=55.555.0.0
UserAgent=Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_4) AppleWebKit/605.1.15 (KHTML, l
Platform=Mac OSX
Application=Browser
City=San Mateo
Country=United States
Subdivision=California
```

LoginDiscoveryHandler Example Implementation

This Apex code example implements the `Auth.LoginDiscoveryHandler` interface. It checks whether the user who is logging in has a verified email or phone number, depending on which identifier wa supplied on the login page. If verified, with `Auth.VerificationMethod.EMAIL` or `Auth.VerificationMethod.SMS`, we send a challenge to the identifier, either the user's email address or mobile device. If the user enters the code correctly on the verify page, the user is redirected to the Experience Cloud site's page specified by the start URL. If the user isn't verified, the user must enter a password to log in. The handler also checks that the email and phone number are unique with this code: `users.size()==1`.

Note

Passwordless login works only with verified methods. You can check the verification status on the User object, for example, with User list view, a report, or the API. Make sure that your solution handles the case where the user doesn't have a verification method. This code example falls back to a password.

The default discoverable login handler checks whether the user entered a valid email address or phone number before redirecting the user to the verification page. If an invalid entry is made, the handler returns an error. Because this behavior is vulnerable to user enumeration attack, make sure that your solution prevents this attack. For example, you can create a dummy page similar to the verification page and redirect the user to the dummy page when invalid user identifier is entered. Also, use generic error messages to avoid providing additional information.

```
global class AutocreatedDiscLoginHandler1535377170343 implements Auth.LoginDiscoveryHandl

global PageReference login(String identifier, String startUrl, Map<String, String> reque
    if (identifier != null && isValidEmail(identifier)) {
        // Search for user by email.
        List<User> users = [SELECT Id FROM User WHERE Email = :identifier AND IsActive =
        if (!users.isEmpty() && users.size() == 1) {
            // User must have a verified email before using this verification method.
            // We cannot send messages to unverified emails.
            // You can check if the user's email verified bit set and add the
            // password verification method as fallback.
            List<TwoFactorMethodsInfo> verifiedInfo = [SELECT HasUserVerifiedEmailAddres
            if (!verifiedInfo.isEmpty() && verifiedInfo[0].HasUserVerifiedEmailAddress =
                // Use email verification method if the user's email is verified.
                return discoveryResult(users[0], Auth.VerificationMethod.EMAIL, startUrl
            } else {
                // Use password verification method as fallback
                // if the user's email is unverified.
                return discoveryResult(users[0], Auth.VerificationMethod.PASSWORD, startU
            }
        } else {
            throw new Auth.LoginDiscoveryException('No unique user found. User count=' +
        }
    }
    if (identifier != null) {
        String formattedSms = getFormattedSms(identifier);
        if (formattedSms != null) {
            // Search for user by SMS.
            List<User> users = [SELECT Id FROM User WHERE MobilePhone = :formattedSms AN
            if (!users.isEmpty() && users.size() == 1) {
                // User must have a verified SMS before using this verification method.
                // We cannot send messages to unverified mobile numbers.
                // You can check if the user's mobile verified bit is set or add
                // the password verification method as fallback.
                List<TwoFactorMethodsInfo> verifiedInfo = [SELECT HasUserVerifiedMobileN
                if (!verifiedInfo.isEmpty() && verifiedInfo[0].HasUserVerifiedMobileNumb
                    // Use SMS verification method if the user's mobile number is verifi
                    return discoveryResult(users[0], Auth.VerificationMethod.SMS, startU
                } else {
                    // Use password verification method as fallback if the user's
                    // mobile number is unverified.
                    return discoveryResult(users[0], Auth.VerificationMethod.PASSWORD, s
                }
            } else {
                throw new Auth.LoginDiscoveryException('No unique user found. User count
            }
        }
    }
    if (identifier != null) {
        // You can customize the code to find user via other attributes,
        // such as SSN or Federation ID.
    }
    throw new Auth.LoginDiscoveryException('Invalid Identifier');
}
private boolean isValidEmail(String identifier) {
    String emailRegex = '^[a-zA-Z0-9._|\\\\%#~`=?&/$^*!}{+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{
    // source: https://www.regular-expressions.info/email.html
    Pattern EmailPattern = Pattern.compile(emailRegex);
    Matcher EmailMatcher = EmailPattern.matcher(identifier);
    if (EmailMatcher.matches()) { return true; }
    else { return false; }
}
private String getFormattedSms(String identifier) {
    // Accept SMS input formats with 1- or 2-digit country code,
    // 3-digit area code, and 7-digit number.
    // You can customize the SMS regex to allow different formats.
    String smsRegex = '^(\\+?\\d{1,2}?[\\s-])?(\\(?\\d{3}\\)?[\\s-]?\\d{3}[\\s-]?\\d{4})
    Pattern smsPattern = Pattern.compile(smsRegex);
    Matcher smsMatcher = SmsPattern.matcher(identifier);
    if (smsMatcher.matches()) {
        try {
```

```
            return null;
        }
    } else { return null; }
}
private PageReference getSsoRedirect(User user, String startUrl, Map<String, String> req
    // You can look up to check whether the user should log in with
    // SAML or an Auth Provider and return the URL to initialize SSO.
    return null;
}
private PageReference discoveryResult(User user, Auth.VerificationMethod method, String
    // Only users with an External Identity or community license can log in
    // using Site.passwordlessLogin. Use getSsoRedirect to let your org employees
    // log in to an Experience Cloud site.
    PageReference ssoRedirect = getSsoRedirect(user, startUrl, requestAttributes);
    if (ssoRedirect != null) {
        return ssoRedirect;
    } else {
        if (method != null) {
            List<Auth.VerificationMethod> methods = new List<Auth.VerificationMethod>();
            methods.add(method);
            PageReference pwdlessRedirect = Site.passwordlessLogin(user.Id, methods, sta
            if (pwdlessRedirect != null) {
                return pwdlessRedirect;
            } else {
                throw new Auth.LoginDiscoveryException('No Passwordless Login redirect U
            }
        } else {
            throw new Auth.LoginDiscoveryException('No method found');
        }
    }
}
}
```

Code Example: Filter Login Discovery Users by Profile

Your production org can have multiple users with the same verified email address and mobile number. But your customers must have unique ones. To address this problem, you can add a few lines of code that filters users by profile to ensure uniqueness. This code example handles users with the External Identity User profile, but can be adapted to support other use cases. For example you can modify the first line of code to address users with other user licenses or criteria.

Login Discovery is available with the following user licenses: Customer Community, Customer Community Plus, External Identity, Partner Community, and Partner Community Plus. It depends on which profiles have access to your Experience Cloud site.

```
global class AutocreatedDiscLoginHandler1551301979709 implements Auth.LoginDiscoveryHand

global PageReference login(String identifier, String startUrl, Map<String, String> reque
    if (identifier != null && isValidEmail(identifier)) {
        // Ensure uniqueness by profile
        Profile p = [SELECT id FROM profile WHERE name = 'External Identity User'];
        List<User> users = [SELECT Id FROM User WHERE Email = :identifier AND IsActive =
        if (!users.isEmpty() && users.size() == 1) {
            // User must have verified email before using this verification method. We ca
            // You can check if the user has email verified bit on and add the password
            List<TwoFactorMethodsInfo> verifiedInfo = [SELECT HasUserVerifiedEmailAddres
            if (!verifiedInfo.isEmpty() && verifiedInfo[0].HasUserVerifiedEmailAddress =
                // Use email verification method if the user's email is verified.
                return discoveryResult(users[0], Auth.VerificationMethod.EMAIL, startUrl
            } else {
                // Use password verification method as fallback if the user's email is u
                return discoveryResult(users[0], Auth.VerificationMethod.PASSWORD, start
            }
        } else {
            throw new Auth.LoginDiscoveryException('No unique user found. User count=' +
        }
```

```
            List<User> users = [SELECT Id FROM User WHERE MobilePhone = :formattedSms AN
            if (!users.isEmpty() && users.size() == 1) {
                // User must have verified SMS before using this verification method. We
                // You can check if the user has mobile verified bit on or add the passw
                List<TwoFactorMethodsInfo> verifiedInfo = [SELECT HasUserVerifiedMobileN
                if (!verifiedInfo.isEmpty() && verifiedInfo[0].HasUserVerifiedMobileNumb
                    // Use SMS verification method if the user's mobile number is verifi
                    return discoveryResult(users[0], Auth.VerificationMethod.SMS, startU
                } else {
                    // Use password verification method as fallback if the user's mobile
                    return discoveryResult(users[0], Auth.VerificationMethod.PASSWORD, s
                }
            } else {
                throw new Auth.LoginDiscoveryException('No unique user found. User count
            }
        }
    }
    if (identifier != null) {
        // You can customize the code to find user via other attributes, such as SSN or
    }
    throw new Auth.LoginDiscoveryException('Invalid Identifier');
}

private boolean isValidEmail(String identifier) {
    String emailRegex = '^[a-zA-Z0-9._|\\\\%#~`=?&/$^*!}{+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{
    // source: https://www.regular-expressions.info/email.html
    Pattern EmailPattern = Pattern.compile(emailRegex);
    Matcher EmailMatcher = EmailPattern.matcher(identifier);
    if (EmailMatcher.matches()) { return true; }
    else { return false; }
}

private String getFormattedSms(String identifier) {
    // Accept SMS input formats with 1 or 2 digits country code, 3 digits area code and
    // You can customize the SMS regex to allow different formats
    String smsRegex = '^(\\+?\\d{1,2}?[\\s-])?(\\(?\\d{3}\\)?[\\s-]?\\d{3}[\\s-]?\\d{4})
    Pattern smsPattern = Pattern.compile(smsRegex);
    Matcher smsMatcher = SmsPattern.matcher(identifier);
    if (smsMatcher.matches()) {
        try {
            // Format user input into the verified SMS format '+xx xxxxxxxxx' before DB
            // Append US country code +1 by default if no country code is provided
            String countryCode = smsMatcher.group(1) == null ? '+1' : smsMatcher.group(1
            return System.UserManagement.formatPhoneNumber(countryCode, smsMatcher.group
        } catch(System.InvalidParameterValueException e) {
            return null;
        }
    } else { return null; }
}

private PageReference getSsoRedirect(User user, String startUrl, Map<String, String> req
    // You can look up if the user should log in with SAML or an Auth Provider and retur
    return null;
}

private PageReference discoveryResult(User user, Auth.VerificationMethod method, String
    //Only users with an External Identity or community license can login using Site.pas
    //Use getSsoRedirect to enable your org employees to log in to an Experience Cloud s
    PageReference ssoRedirect = getSsoRedirect(user, startUrl, requestAttributes);
    if (ssoRedirect != null) {
        return ssoRedirect;
    } else {
        if (method != null) {
            List<Auth.VerificationMethod> methods = new List<Auth.VerificationMethod>();
            methods.add(method);
            PageReference pwdlessRedirect = Site.passwordlessLogin(user.Id, methods, sta
            if (pwdlessRedirect != null) {
                return pwdlessRedirect;
            } else {
                throw new Auth.LoginDiscoveryException('No Passwordless Login redirect U
            }
        } else {
```

DID THIS ARTICLE SOLVE YOUR ISSUE?
Let us know so we can improve!

Share your feedback