Authentication Challenges and TLS Chain Validation

An `NSURLRequest` object often encounters an **authentication challenge**, or a request for credentials from the server it is connecting to. The `NSURLSession`, `NSURLConnection`, and `NSURLDownload` classes notify their delegates when a request encounters an authentication challenge, so that they can act accordingly.

> **Important:** The URL loading system classes do not call their delegates to handle request challenges unless the server response contains a `WWW-Authenticate` header. Other authentication types, such as proxy authentication and TLS trust validation do not require this header.

Deciding How to Respond to an Authentication Challenge

If an `NSURLRequest` object requires authentication, the way that the challenge is presented to your app varies depending on whether the request is performed by an `NSURLSession` object, an `NSURLConnection` object, or an `NSURLDownload` object:

- If the request is associated with an `NSURLSession` object, all authentication requests are passed to the delegate, regardless of authentication type.
- If the request is associated with an `NSURLConnection` or `NSURLDownload` object, that object's delegate receives a `connection:canAuthenticateAgainstProtectionSpace:` (or `download:canAuthenticateAgainstProtectionSpace:`) message. This allows the delegate to analyze properties of the server, including its protocol and authentication method, before attempting to authenticate against it. If your delegate is not prepared to authenticate against the server's protection space, you can return `NO`, and the system attempts to authenticate with information from the user's keychain.
- If the delegate of an `NSURLConnection` or `NSURLDownload` object does not implement the `connection:canAuthenticateAgainstProtectionSpace:` (or `download:canAuthenticateAgainstProtectionSpace:`) method and the protection space uses client certificate authentication or server trust authentication, the system behaves as if you had returned `NO`. The system behaves as if you returned `YES` for all other authentication types.

Next, if your delegate agrees to handle authentication and there are no valid credentials available, either as part of the requested URL or in the shared `NSURLCredentialStorage`, the delegate receives one of the following messages:

```
URLSession:didReceiveChallenge:completionHandler:
URLSession:task:didReceiveChallenge:completionHandler:
connection:didReceiveAuthenticationChallenge:
download:didReceiveAuthenticationChallenge:
```

In order for the connection to continue, the delegate has three options:

- Provide authentication credentials.
- Attempt to continue without credentials.
- Cancel the authentication challenge.

To help determine the correct course of action, the `NSURLAuthenticationChallenge` instance passed to the method contains information about what triggered the authentication challenge, how

many attempts were made for the challenge, any previously attempted credentials, the `NSURLProtectionSpace` that requires the credentials, and the sender of the challenge.

If the authentication challenge has tried to authenticate previously and failed (for example, if the user changed his or her password on the server), you can obtain the attempted credentials by calling `proposedCredential` on the authentication challenge. The delegate can then use these credentials to populate a dialog that it presents to the user.

Calling `previousFailureCount` on the authentication challenge returns the total number of previous authentication attempts, including those from different authentication protocols. The delegate can provide this information to the user, to determine whether the credentials it supplied previously are failing, or to limit the maximum number of authentication attempts.

Responding to an Authentication Challenge

The following are the three ways you can respond to the `connection:didReceiveAuthenticationChallenge:` delegate method.

Providing Credentials

To attempt to authenticate, the application should create an `NSURLCredential` object with authentication information of the form expected by the server. You can determine the server's authentication method by calling `authenticationMethod` on the protection space of the provided authentication challenge. Some authentication methods supported by `NSURLCredential` are:

- HTTP basic authentication (`NSURLAuthenticationMethodHTTPBasic`) requires a user name and password. Prompt the user for the necessary information and create an `NSURLCredential` object with `credentialWithUser:password:persistence:`.
- HTTP digest authentication (`NSURLAuthenticationMethodHTTPDigest`), like basic authentication, requires a user name and password. (The digest is generated automatically.) Prompt the user for the necessary information and create an `NSURLCredential` object with `credentialWithUser:password:persistence:`.
- Client certificate authentication (`NSURLAuthenticationMethodClientCertificate`) requires the system identity and all certificates needed to authenticate with the server. Create an `NSURLCredential` object with `credentialWithIdentity:certificates:persistence:`.
- Server trust authentication (`NSURLAuthenticationMethodServerTrust`) requires a trust provided by the protection space of the authentication challenge. Create an `NSURLCredential` object with `credentialForTrust:`.

After you've created the `NSURLCredential` object:

- For `NSURLSession`, pass the object to the authentication challenge's sender using the provided completion handler block.
- For `NSURLConnection` and `NSURLDownload`, pass the object to the authentication challenge's sender with `useCredential:forAuthenticationChallenge:`.

Continuing Without Credentials

If the delegate chooses not to provide a credential for the authentication challenge, it can attempt to continue without one.

- For `NSURLSession`, pass one of the following values to the provided completion handler block:

  `NSURLSessionAuthChallengePerformDefaultHandling` processes the request as though the delegate did not provide a delegate method to handle the challenge.

  `NSURLSessionAuthChallengeRejectProtectionSpace` rejects the challenge. Depending on the authentication types allowed by the server's response, the URL loading class may call this delegate method more than once, for additional protection spaces.

- For `NSURLConnection` and `NSURLDownload`, call `continueWithoutCredentialsForAuthenticationChallenge:` on `[challenge sender]`.

  Depending on the protocol implementation, continuing without credentials may either cause the connection to fail, resulting in a `connectionDidFailWithError:` message, or return alternate URL contents that don't require authentication.

Canceling the Connection

The delegate may also choose to cancel the authentication challenge.

- For `NSURLSession`, pass `NSURLSessionAuthChallengeCancelAuthenticationChallenge` to the provided completion handler block.
- For `NSURLConnection` or `NSURLDownload`, call `cancelAuthenticationChallenge:` on `[challenge sender]`. The delegate receives a `connection:didCancelAuthenticationChallenge:` message, providing the opportunity to give the user feedback.

An Authentication Example

The implementation shown in Listing 6-1 responds to the challenge by creating an `NSURLCredential` instance with a user name and password supplied by the application's preferences. If the authentication has failed previously, it cancels the authentication challenge and informs the user.

Listing 6-1 An example of using the `connection:didReceiveAuthenticationChallenge:` delegate method

```
-(void)connection:(NSURLConnection *)connection
        didReceiveAuthenticationChallenge:(NSURLAuthenticationChallenge *)challenge
{
    if ([challenge previousFailureCount] == 0) {
        NSURLCredential *newCredential;
        newCredential = [NSURLCredential credentialWithUser:[self preferencesName]
                                                   password:[self preferencesPassword]

persistence:NSURLCredentialPersistenceNone];
        [[challenge sender] useCredential:newCredential
               forAuthenticationChallenge:challenge];
    } else {
        [[challenge sender] cancelAuthenticationChallenge:challenge];
        // inform the user that the user name and password
        // in the preferences are incorrect
        [self showPreferencesCredentialsAreIncorrectPanel:self];
    }
}
```

If the delegate doesn't implement `connection:didReceiveAuthenticationChallenge:` and the request requires authentication, valid credentials must already be available in the URL credential storage or must be provided as part of the requested URL. If the credentials are not available or if they fail to authenticate, a `continueWithoutCredentialForAuthenticationChallenge:` message is sent by the underlying implementation.

Performing Custom TLS Chain Validation

In the NSURL family of APIs, TLS chain validation is handled by your app's authentication delegate method, but instead of providing credentials to authenticate the user (or your app) to the server, your app instead checks the credentials that the server provides during the TLS handshake, then tells the URL loading system whether it should accept or reject those credentials.

If you need to perform chain validation in a nonstandard way (such as accepting a specific self-signed certificate for testing), your app must do the following:

- For `NSURLSession`, implement either the `URLSession:didReceiveChallenge:completionHandler:` or `URLSession:task:didReceiveChallenge:completionHandler:` delegate method. If you implement both, the session-level method is responsible for handling the authentication.
- For `NSURLConnection` and `NSURLDownload`, implement the `connection:canAuthenticateAgainstProtectionSpace:` or `download:canAuthenticateAgainstProtectionSpace:` method and return `YES` if the protection space has an authentication type of `NSURLAuthenticationMethodServerTrust`.

  Then, implement the `connection:didReceiveAuthenticationChallenge:` or `download:didReceiveAuthenticationChallenge:` method to handle the authentication.

Within your authentication handler delegate method, you should check to see if the challenge protection space has an authentication type of `NSURLAuthenticationMethodServerTrust`, and if so, obtain the `serverTrust` information from that protection space.

For additional details and code snippets (based on `NSURLConnection`), read Overriding TLS Chain Validation Correctly.
