Security and Privacy Considerations

WebAuthn Starter Kit Security Considerations

When deploying a service based on the WebAuthn Starter kit, there are a number of tangential security elements to be aware of. While outside the scope included in the WebAuthn Starter Kit and WebAuthn authentication flows, aspects of the greater environment in an actual deployment may introduce potential avenues to bypass the protections therein.

Cross-Origin Resource Sharing

Cross-origin resource sharing (CORS) is a browser security feature that restricts cross-origin HTTP requests that are initiated from scripts running in the browser. The WebAuthn Starter kit is created to offer an abundance of flexibility in how it can be deployed to other systems. As such, the Cross-Origin Resource Sharing (CORS) header defaults to everywhere. In an actual deployment, this should obviously be restricted to only the needed origins.

For typical deployments, the CORS header would include only `companyname.com` and `www.companyname.com` at a minimum for a javascript web client making requests to the Lambda. For further reference, refer to AWS' documentation on this subject: Enabling CORS for a REST API resource

Account Recovery

When deploying WebAuthn, one of the key flows to consider is the user’s recovery of their account should their authenticator become lost or stolen. There are multiple avenues to addressing this, from policies such as requiring multiple authenticators per account, to recovery authentication flows built into a service. The correct approach for Account Recovery is highly dependent on the security and sensitivity of the data protected. Regardless, it is important that the account recovery flow is well protected; malicious actors attempting to gain access to protected accounts will bypass a strong primary form of authentication if they can utilize the lesser-protected account recovery functions instead.

The WebAuthn Starter kit offers a very basic account recovery flow, with the intent of demonstrating how to enable such a flow into a WebAuthn integration. For deployments, we recommend using an existing account recovery option is already in place for a service, if one already exists.

For new services, the Digital Identity Guidelines published by the National Institute of Standards and Technology provide an excellent resource on mitigating the risks to account recovery.

WebAuthn Standard Privacy Considerations

The WebAuthn standard level 2 editor's draft includes the section §14. Privacy Considerations. This section describes in depth the privacy aspects of WebAuthn.

For the WebAuthnKit in particular, the privacy concerns addressed in section §14.6.2. Username Enumeration in conjunction with §14.6.3. Privacy leak via credential IDs need to be considered.

The most relevant part from section §14.6.2. Username Enumeration is copied below:

“While initiating a registration or authentication ceremony, there is a risk that the WebAuthn Relying Party might leak sensitive information about its registered users. For example, if a Relying Party uses e-mail addresses as usernames and an attacker attempts to initiate an authentication ce remony for "alex.p.mueller@example.com" and the Relying Party responds with a failure, but then successfully initiates an authentication ceremony for "j.doe@example.com", then the attacker can conclude that "j.doe@example.com" is registered and "alex.p.mueller@example.com" is not.

For authentication ceremonies:

- If, when initiating an authentication ceremony, there is no account matching the provided username, continue the ceremony by invoking `navigator.credentials.get()` using a syntactically valid `PublicKeyCredentialRequestOptions` object that is populated with plausible imaginary values.

This approach could also be used to mitigate information leakage via `allowCredentials`; see §13.5.6 Unprotected account detection and §14.6.3 Privacy leak via credential IDs."

The most relevant part from section §14.6.3. Privacy leak via credential IDs is copied below:

"This privacy consideration applies to Relying Parties that support authentication ceremonies with a non-empty `allowCredentials` argument as the first authentication step. For example, if using authentication with server-side credentials as the first authentication step.

In order to prevent such information leakage, the Relying Party could for example:

- Perform a separate authentication step, such as username and password authentication or session cookie authentication, before initiating the WebAuthn authentication ceremony and exposing the user's credential IDs.
- Use client-side discoverable credentials, so the `allowCredentials` argument is not needed.

If the above prevention measures are not available, i.e., if `allowCredentials` needs to be exposed given only a username, the Relying Party could mitigate the privacy leak using the same approach of returning imaginary credential IDs as discussed in §14.6.2 Username Enumeration."

Solutions in the WebAuthn Starter Kit

This section describes the solutions that are implemented in the WebAuthn Starter Kit to mitigate the privacy leaks.

Use client-side discoverable credentials

The Recommendation in section §14.6.3. Privacy leak via credential IDs: "Use client-side discoverable credentials, so the allowCredentials argument is not needed." In short, the allowCredentials argument should be left empty when dealing with client-side discoverable credentials.

This recommendation is relevant for the WebAuthnKit for the User-Verified flow, in the case where a FIDO authenticator with client-side discoverable credentials is used. The WebAuthnKit returns an empty list of allowCredentials, preventing a malicious actor from identifying which accounts are secured with WebAuthn client-side discoverable credentials and which are not, and potentially easier to attack.

Perform a separate authentication step

The Recommendation in section §14.6.3. Privacy leak via credential IDs: "Perform a separate authentication step, such as username and password authentication or session cookie authentication, before initiating the WebAuthn authentication ceremony and exposing the user's credential IDs."

This recommendation is relevant for the WebAuthnKit for the Server-Verified flow. This is also supported by the WebAuthnKit for the Server-Verified flow, where the user is first authenticated with username and Server-Verified PIN, before the list of user's credential IDs is exposed. This likewise prevents malicious actors from attempting to mine for data on registered accounts.

Return imaginary credential IDs

The Recommendation in section §14.6.2. Username Enumeration: "If, when initiating an authentication ceremony, there is no account matching the provided username, continue the ceremony by invoking `navigator.credentials.get()` using a syntactically valid `PublicKeyCredentialRequestOptions` object that is populated with plausible imaginary values."

This recommendation is relevant for the WebAuthnKit for the Server-Verified flow. - however, this is not yet supported by the WebAuthnKit for the Server-Verified flow. Currently, the Server-Verified PIN authentication step is supported by the WebAuthnKit, which to some extent mitigates the risk of an attacker trying to list the credentials. To be fully protected against this risk, changes need to be implemented at the WebAuthn server level, which in turn can be integrated into the WebAuthnKit back-end.