OAuth 2.0 Token Binding

Brian Campbell
Michael B. Jones
John Bradley

IETF 97
Seoul
November 2016

https://tools.ietf.org/html/draft-ietf-oauth-token-binding-01

Why?

- Specify a means of using Token Binding with OAuth (& OpenID Connect) to to defeat replay of stolen tokens
  - Refresh tokens
  - (ID Tokens)
  - Access tokens
  - Authorization Codes

Status

- After Berlin draft-jones-oauth-token-binding-00 adopted as starting point for WG draft
- Unchanged to initial working group version draft-ietf-oauth-token-binding-00

- draft-ietf-oauth-token-binding-01
  - Changed Token Binding for access tokens to use the Referred Token Binding ID vs. an authorization request parameter
  - Defined Protected Resource Metadata value.
  - Changed to use the more specific term "protected resource" instead

Quick Token Binding Overview

- Uses a public-private key pair generated by the client to sign TLS exported keying material and create long-lived TLS binding
- draft-ietf-tokbind (TBNEGO)
  - -negotiation-05
    - TLS extension for token binding protocol negotiation
  - -protocol-10 (TBPROTO)
    - Token Binding protocol message format
      - provided & referred types
  - -https-06 (HTTPSTB)
    - Embedding token binding messages in HTTPS
      - Sec-Token-Binding request header
      - Include-Referred-Token-Binding-ID response header

Token Binding for Refresh Tokens

- Section 2 of draft-ietf-oauth-token-binding-01
- Straightforward (like binding a cookie)
- There’s only the Client and AS
- When issuing an RT, AS binds it to the provided Token Binding ID from the client
- When presented with an RT, AS checks its bound Token Binding ID against the provided TB from the client
- Transparent to the client

Representing Token Binding in JWTs & ID Tokens

- New RFC 7800 JWT Confirmation Method member, “tbh”
  - SHA-256 hash of a Token Binding ID in an ID Token
  - Defined in OpenID Connect Token Bound Authentication (http://openid.net/specs/openid-connect-token-bound-authentication-1_0.html)

```
{
  "iss": "https://as.example.com",
  "aud": "https://resource.example.com",
  "sub": "user@example.com",
  "exp": 1478891626,
  "cnf":{
    "tbh": ”8ESC_3r1ACCGp2qiLOf48BWCTjpBnhm-QOyzJxhyLTC"
  }
}
```

Token Binding for Access Tokens

- Section 3 of draft-ietf-oauth-token-binding-01
- Binds the access token to the token binding key used by the client in the TLS connection to the protected resource
- When issuing an AT the AS binds it to the referred Token Binding ID presented at the,
  - Token endpoint (code, refresh, and all other grants)
  - Authorization endpoint (implicit)
- Protected resource validates by comparing the Provided Token Binding ID to the Token Binding ID for the access token

Referred Token Binding ID

- Conceptually the *right* approach but
  - No redirect occurs between the protected resource and the authorization server
  - Some allowance for native applications in HTTPSTB but “applications MUST only convey Token Binding IDs to other servers if the server associated with a Token Binding ID explicitly signals to do so, e.g., by returning an Include-Referred-Token-Binding-ID HTTP response header field”
    - Get that text changed
    - Interpret that text very liberally
    - Add an explicit signal (maybe a new auth-param with the WWW-Authenticate Response Header Field from RFC 6750)
  - May still prove cumbersome in some situations
    - Native app using different code path for token endpoint and API access
    - Clustered web server clients
    - Etc.
  - HTTPSTB has a SHOULD for an eTLD+1 scoping requirement

Token Binding for Authorization Codes

- Work outstanding to be added to the draft
- Two flavors:
  - Bind to the Token Binding ID the native client uses to resolve the code at the token endpoint
  - Bind to the Token Binding ID the browser uses to deliver the code to a web server client
    - Defeats cut-and-paste replay
- Is a double binding necessary?

Authorization Code Binding Straw-man

- Bind to the Token Binding ID the native client uses to resolve the code at the token endpoint
  - code_challenge=BASE64URL(SHA256(Provided Token Binding ID between client and AS token endpoint))
  - code_challenge_method=tbs256
  - code_verifier=provided (and use the value of the provided Token Binding ID)
- Bind to the Token Binding ID the browser uses to deliver the code to a web server client
  - code_challenge=referred (use the value of the referred Token Binding ID)
  - code_challenge_method=referred_tb
  - code_verifier=BASE64URL(Provided Token Binding ID between browser and Client's redirect URI)

Token Binding Metadata

- Client
  - client_access_token_token_binding_supported (Boolean)
  - client_refresh_token_token_binding_supported (Boolean)
- Authorization Server
  - as_access_token_token_binding_supported (Boolean)
  - as_refresh_token_token_binding_supported (Boolean)
- Protected Resource
  - resource_access_token_token_binding_supported (Boolean)

Phasing in Token Binding & Preventing Downgrade Attacks

- Token Binding won’t bind if not all participants support it
  - ‘context-dependent deployment choice whether to allow interactions to proceed’ (recommended in the general case to allow)
- Downgrade: if all participants support it but one doesn’t use it, ‘likely evidence of a downgrade attack […] authorization SHOULD be aborted with an error.’
  - It’s more subtle than that, mismatch in supported key parameters types would lead to the same situation
  - Supported key parameters types vs Boolean in metadata?
  - Metadata for class of Client apps might not be able to accurately convey
  - AS may not know the resource(s)

(Known) Next Steps

- (somehow) resolve conflict in HTTPSTB with explicit signaling needed to reveal the referred Token Binding
- Add binding for authorization codes
- Flesh out or back off of metadata and downgrade detection