Authorization Code Log File Attack

Will Bartlett, Mike Jones, Pieter Kasselman

Purpose

This document describes how an attacker can obtain Access and Refresh Tokens even if PKCE and draft-ietf-oauth-dpop-04 are used. The attack can be mitigated by binding the DPoP public key to the Authorization Code. This can be achieved by adding a new PKCE method that uses the JSON Web Key (JWK) Thumbprint of the DPoP key as a "code_challenge" method or by supplying the JWK Thumbprint as an additional parameter in the Authorization Request.

Attack Description

VPNs, network proxies, and applications may indiscriminately log network traffic, including Authorization Codes and PKCE Code Verifiers. Attackers may gain access to these logs and use the Authorization Code and PKCE Code Verifiers to obtain Access and Refresh Tokens, bound to a DPoP key that they generated and control. Figure 1 below shows a step-by-step flow of the attack:

1. The Client initiates an Authorization Request and a PKCE Code (using S256), as defined in RFC 7636.
2. The Client receives an Authorization Response, including the Authorization Code (AC).
3. The Client sends a Token Request, presenting the Authorization Code, PKCE Verifier and, a DPoP key and proof.
    a. The Authorization Code and PKCE Verifier is logged (by the application or by a VPN, proxy, or other participant).
4. The Authorization Server issues sender-constrained Access and Refresh Tokens to the Client.
5. The attacker accesses the log files that contains the Authorization Code and the PKCE code verifier and exfiltrates the Authorization Code and the PKCE code verifier to an environment it controls to execute the remainder of the attack.
6. The attacker sends a Token Request using the "client_id", Authorization Code, and PKCE Verifier it obtained from the log files. It generates its own DPoP key (DPoP_Key_B)and calculates the corresponding DPoP proof and sends it to the Authorization Server, along with the DPoP proof.
7. The Authorization Server does not enforce one-time use of the Authorization Code. It checks the "client_id", verifies the PKCE code verifier, and then binds the Access and Refresh Token to the attacker's DPoP key and returns them to the attacker.

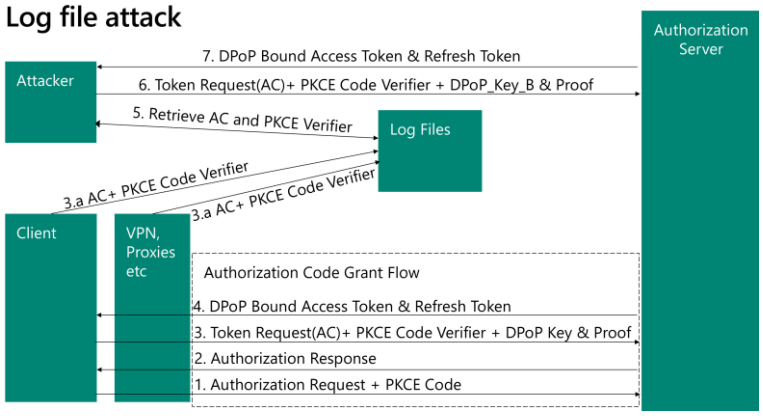

Figure 1: Log File Attack

Variation

In a variation of this attack, the Authorization Server enforces one-time use, but the attacker interrupts the communications at step 3 to prevent the Authorization Server from receiving the Token Request that includes the Authorization Code or the PKCE Code Verifier, although both are still written to the logs.

Mitigations

The attack may be mitigated by binding the Authorization Code to the DPoP key. This binding can be achieved in two ways. A JWK Thumbprint of the DPoP Key can be presented using a new type of PKCE method or it can be included as a new parameter in the Authorization Request. In both cases, the hash of the DPoP key is used for efficiency purposes.

This mitigation is effective even when the Authorization Server is unable to guarantee one-time use of the Authorization Code.

DPoP PKCE Method

The JWK Thumbprint may be presented through a new PKCE DPoP method. The PKCE DPoP method uses the JSON Web Key (JWK) Thumbprint as the "code_challenge" value. Logically, the code verifier value is the proof-of-possession public key. However, the `code_verifier` parameter is set to the constant value "DPoP" with this PKCE method because the proof-of-possession public key is already being sent in the DPoP proof. When a token request is received, the authorization server computes the JWK Thumbprint of the proof-of-possession public key in the DPoP proof and verifies that it matches the code challenge. If they do not match, it rejects the request. The Authorization Server verifies the DPoP Proof as part of regular DPoP processing, thereby confirming that the Client controls the matching private key.

Additional Parameter

In this approach, the JWK Thumbprint of the Client's DPoP key is sent to the Authorization Server as an additional parameter to the Authorization Request, along with the PKCE Code (using the "S256" method). When the Client performs a Token Request, it provides the Authorization Code, DPoP Key, and DPoP Proof. The Authorization Server verifies the DPoP Proof, compares the hash of the DPoP public key with the JWK Thumbprint presented in the original Authorization Request along with verification of the PKCE Code Verifier, and performs other standard checks before issuing the sender constrained Access and Refresh Tokens.

Discussion

If the Authorization Code, PKCE Code Verifier and DPoP proof are logged and an attacker access them, the attacker may still present them and receive an Access and Refresh Token, but they will be bound to the Client's DPoP key, and the attacker will not be able to use them.

If the attacker tries to substitute the DPoP Proof with their own, generated with DPoP_Key_B, the Authorization Server will decline issuing any tokens since the DPoP proof presented by the attacker cannot be verified with the DPoP key presented to it in the Authorization Request.

As an additional precaution, the Authorization Server can invoke the optional DPoP Nonce capability, forcing the attacker to generate a fresh signature. Since the attacker does not control the key that was sent in the original Authorization Request, they are unable to generate a valid DPoP proof and consequently can't obtain Access or Refresh Tokens.

Note that the attacker is unable to substitute the DPoP key presented in the original Authorization Request since they don't control the session for the original Authorization Request.