GNAP Meeting Interim 2021-06

draft-ietf-gnap-core-protocol-05
(and changes)

June 15, 2021

Justin Richer • Aaron Parecki • Fabien Imbault

Agenda

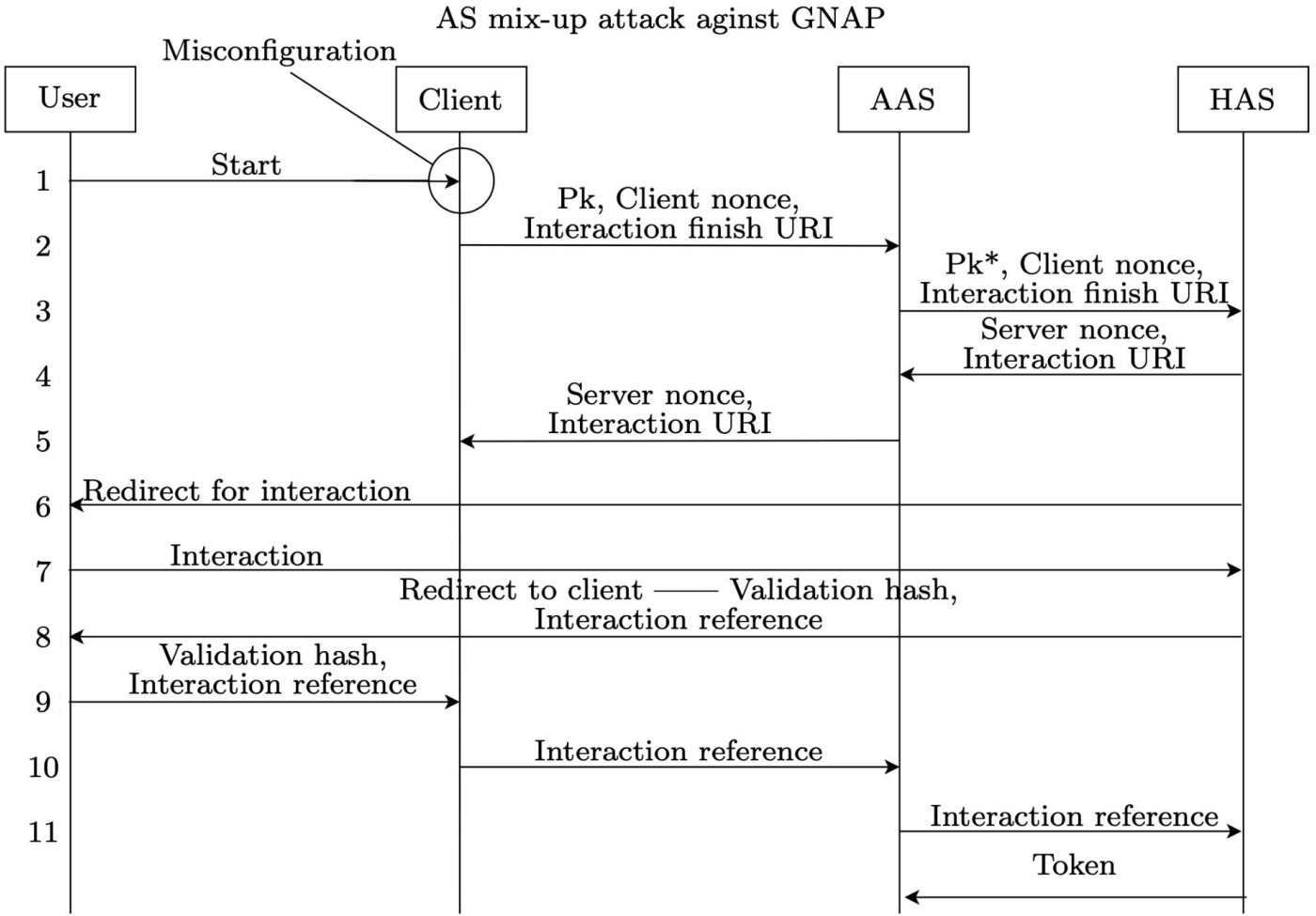
- Core draft update
- Mix-up attack
- Signature methods
- What topics to focus on before IETF 111?

Functional Changes

- Added “privileges” field to access request
- Moved RS-first discovery back to core
  - Turns out it was client-facing all along!

GNAP Mix-Up Attack

- Related to the OAuth 2 “Mix Up” attack
- Mitigation already proposed (extend interaction hash)
- How it works:
  - Attacker gets uncompromised client (UC) to talk to attacker AS (AAS)
  - AAS acts as a different client instance to home AS (HAS)
  - AAS proxies UC’s request to HAS to start transaction and kick off interaction
  - User interacts with HAS and approves AAS
  - UC gives reference back to AAS, AAS gets token
- How it’s different from OAuth 2
  - Client requests are bound to keys instead of bare secrets: no impersonation on the wire
  - Access token is (normally) bound to a key


AS mix-up attack aginst GNAP
Misconfiguration
User
Client
AAS
HAS
1
Start
2
Pk, Client nonce,
Interaction finish URI
3
Pk*, Client nonce,
Interaction finish URI
4
Server nonce,
Interaction URI
5
Server nonce,
Interaction URI
6
Redirect for interaction
7
Interaction
8
Redirect to client —— Validation hash,
Interaction reference
9
Validation hash,
Interaction reference
10
Interaction reference
11
Interaction reference
Token

Attack Steps

1. UC is a client of AAS, and might also be a client of HAS. User wants to authorize at HAS but tells UC to use AAS.
2. UC starts a request at AAS, signed with UC's key. AAS is imitating HAS.
3. AAS forwards UC's request parameters (Client nonce, interaction finish URI) to HAS, but signed with AAS's key.
4. HAS responds with an interaction start URL and server nonce to AAS
5. AAS forwards the interaction start URL and server nonce to UC
6. (Note) HAS is functionally telling the user to show up and interact, but doesn't realize that the request is being proxied by AAS from UC in this way.
7. UC launches interaction start url, which is a function of HAS
8. HAS returns the verification hash and interaction reference to UC
9. UC validates the hash (which is correct) and sends the interaction reference to AAS
10. AAS forwards the interaction reference to HAS
11. AAS receives an access token for calling an RS protected by HAS. The client receives no access token.

Proposed Mitigation

- Add the grant endpoint URL to the interaction hash calculation
  - Known to client instance
  - Known to HAS (and its interaction elements)
  - Known to AAS but can’t be modified or substituted
- Against this attack:
  - HAS uses its own URL to generate hash
  - UC uses the AAS URL to generate validation hash
  - UC hash validation fails and attack stops before interaction reference is presented to AAS
- Similar to OAuth 2 “iss” return parameter, but cryptographically bound

Mitigation discussion

- Redirect-based protocols are inherently phishable
- Methods without interaction “finish” susceptible to similar phishing attacks during polling periods
- Attack is made easier by dynamic clients but possible even with static clients
  - AAS impersonates UC to the user
  - Attacker gets UC to talk to AAS in the first place but convinces user that they’re using HAS

Signature Methods

- Proposal to keep:
  - MTLS
  - HTTP Message Signatures
- Proposal to drop:
  - OAuth PoP
  - DPoP
- Proposal for discussion:
  - Attached JWS
  - Detached JWS

Key Rotation

- Access tokens are bound to keys
  - We allow rotation of the token value at client instance request...
  - Should we allow rotation of the key also?
- Grant transactions are also bound to keys
  - Specifically: the continuation access token is bound to a key
  - The key is initially the client instance’s key
  - Should the client be able to rotate this key separately?
- Some client instances have registered keys
  - What happens when a client’s registered key rotates?

Goals for IETF 111

- What’s next?