Token Theft and Protection

Hirsch Singhal - @hpsin
IETF 114
Anatomy of an attack

1. Get tokens en masse
   Break into a cloud provider or popular integration

2. Target victim codebases
   Read access is all that’s needed

3. Find additional secrets
   AWS keys, client secrets, long-lived tokens

4. Move sideways
   Move from codebase into infrastructure for the real attack
Tokens you can lose

- Personal Access Tokens (PATs)
  Ad-hoc tokens for testing and quick scripts. Sometimes expire, sometimes don’t.

- API Keys
  User-less, client-less key that protects a resource

- Access Tokens
  OAuth-style, rotating, scoped.
Weaknesses

None sender-constrained
Bearer throughout - no auth of the actor to use them.

Most long-lived
Once stolen, must be revoked to remove access.

Most over-permissioned
Too much privilege on too many resources
Advice for app developers

- Encrypt at rest
  If an attacker dumps your DB, make it harder to use.

- Use expiring tokens
  Make token theft a race, and make clean up later timebound.

- Enable secret scanning
  And clean that git history

- Don’t have credentials to lose!
  Use workload federation so that your infrastructure is the authZ, or at least HSM for keys
Service providers discussion

What can token issuers do to make this better?
What can we do today?

- **IP allow-lists for apps**
  - Primitive binding of tokens to known-good locations

- **Limit non-expiring tokens**
  - Make bad patterns hard to implement

- **Support workload federation**
  - Let developers ditch creds and go cross-cloud

- **Partner with secret scanning teams**
  - Register your tokens for detection
Confidently identifiable tokens

Before

je5WG\text{Gi23lgk84GEPQglwafj3slgk2lgiwhio8rgk}

842b9e8fb032869e88b653fc4df0786240ae6174209b8505c2f9e228a21

After

prefix

\text{ghp_iJxyu4JkSaVUS1EVBmaok0YAl56uLr3i0Y7B}

32-bit checksum

\text{dop_v1_ae5067bb5c1d3bcb1f9e580f7a8dd56186f27791101ccc32b}

High entropy (randomness)

https://docs.github.com/en/developers/overview/secret-scanning-partner-program
And then... token binding

How’s that going?
dPoP perceived status

Prioritizing device/client binding over channel binding given current threats

Opinion: 8705 is DoA for us to implement

Mobile apps
App identity, strong isolation

Desktop apps
No strong app identity, but some OS/IdP can device bind access tokens

Confidential clients
No profile to protect tokens stolen from confidential clients. But seems reasonable.

Web apps
No browser support or strong device binding.
What’s our risk profile?

Assume attackers can lift tokens. We should require persistence and infrastructure compromise to succeed.
Concerns and gotchas

Cross-party referred binding
Securing auth code -> bound cookie without a weak link

HSM/TPM rate limits
Not bad for native apps. Lethal for high-activity confidential client backends

Shared-execution environments
How do you bind tokens that are supposed to be shared to other parties, like in CI or Actions?

Privacy
“Durably identify devices over time” is explicitly bad for some
So what next?

Confidential client profile for dPoP

Browser support for dPoP key protection

Guidance and standards for referred binding
Thank you