sgnl

TRANSACTION TOKENS

Identity and Authorization for Microservices

Presented by

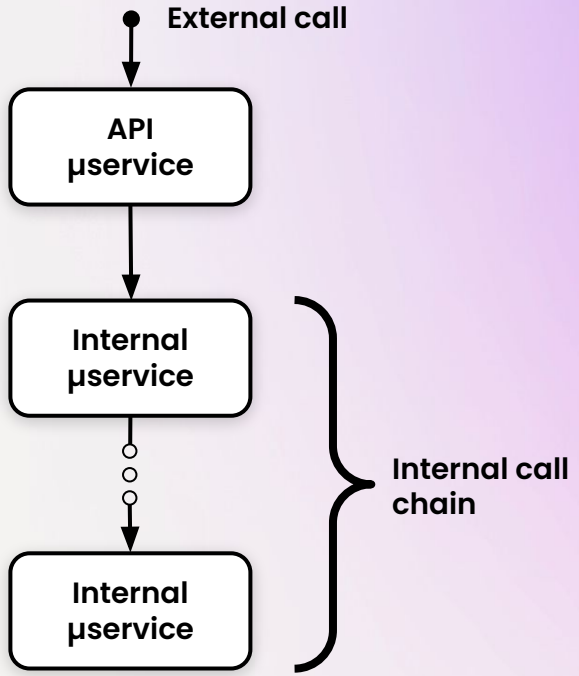
Atul Tulshibagwale CTO, SGNL.ai

@zirotrust

tulshi

Why Transaction Tokens?

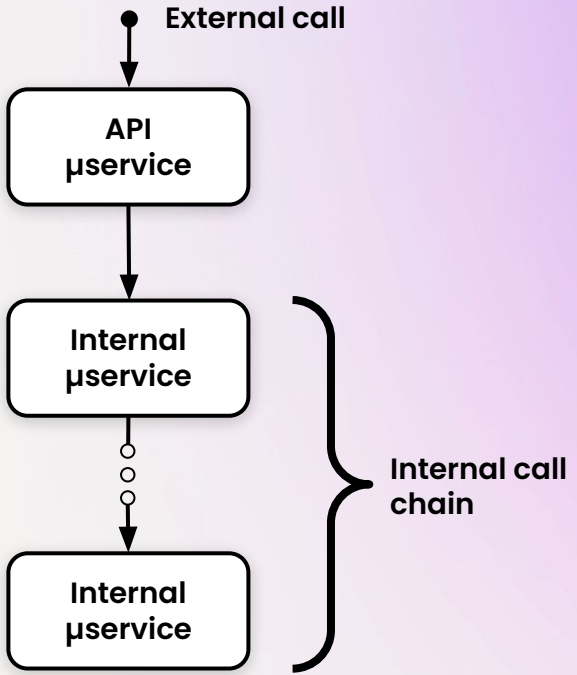
- Microservices architectures result in call chains to service a single external call
- API microservice logically encapsulates any network infrastructure layers
- Calls are short-lived
- Batch jobs can be thought of as a sequence of external calls made by the robotic principal of the batch job. Each such call is short-lived.



sgnl

Why Transaction Tokens?

- Individual microservices in a VPC may be compromised
  - Software supply chain attacks
  - VPC compromise
  - Other attacks
- Attacks can result in
  - User impersonation
  - Arbitrary method invocation



sgnl

Microservices Need More Security

- Implicit Trust
- Service-to-Service Trust
- User Trust
- Assured Context

We're Here :O

We Need to Be Here

sgnl

Microservices Attack Vectors

Privileged User Compromise

(PU Compromise)

- Attacker hijacks identity of someone with admin access
  - Credential compromise
  - Session hijacking
- Attacker makes spurious calls from within the VPC
- Attacker inserts their service in VPC
- Equivalent of RCE in the cloud

Malicious Insider

- Privileged Insider acts like attacker
- Has deep knowledge of the environment
- Otherwise similar to privileged user compromise
- Sometimes there are just “Curious Insiders”

sgnl

SBOM Compromise

- Software supply chain is compromised
- Attacker code “calls home” from your VPC
- Can make spurious calls, though may not run new microservices

sgnl

Microservices Trust Models

Signing and encryption provide security, privacy and integrity, but...

Trust is fundamental to signing and encryption

sgnl

How is Trust Established?

- Using signatures or TLS (including mutual TLS)
- Verifying signatures or trusting a TLS peer requires public keys
- So, public key distribution is central to trust
- Attacker hijacks identity of someone with admin access
    - Embedded “well-known” root keys

sgnl

Implicit Trust in Microservices

- “You’re In the VPC, therefore you’re trusted”
- No security, privacy or integrity protection
- Weakest protection against curious insiders
- VPC compromise is catastrophic

sgnl

Service-to-Service Trust

- “Identify, then trust the Service”
- Still exploitable by curious insiders
    - Requires ability to call APIs in VPC
- PU compromise attacks are still undeterred

SPIFFE

- Defines basic concepts of service-to-service trust
- SPIRE implements key distribution for SPIFFE
- Most common way to implement s-to-s Trust

What is User Trust?

Identity of the user initiating the call is assured

sgnl

Benefits of User Trust

- **Mitigates:** SBOM and PU compromise and malicious insider attacks
- But attackers can still change call context (call parameters)

sgnl

Assured Context

Assures user identity and call context

sgnl

Assured Context Benefits

- **Mitigates:** SBOM and PU compromise and malicious insider attacks
- Attackers cannot even change parameters

Transaction Tokens (TraTs)

Short-lived signed JWTs that assure call context and user identity

sgnl

What is a Transaction Token?

- Short-lived signed JWT
- Uniquely identifies a call chain
- Asserts context that needs to be preserved in a call chain
  - User identity
  - Transaction identifier
  - Originator information
  - Purpose
  - Transaction Context

Benefits of TraTs

- TraTs limit damage by
  - Preserving context immutably throughout a call chain
  - Services can assert that they have processed a call to downstream services
- Context may include
  - Identity of user or robotic principal initiating the external call
  - Parameters provided by the external caller
  - Other data that needs to be preserved in the call chain

sgnl

Benefits of TraTs

- **Mitigates:** SBOM and PU compromise and malicious insider attacks
- Attackers cannot even change parameters

But… they could still bypass services

sgnl

TraT Structure

```
{
  "iss": "https://trust-domain.example/trat-service",
  "iat": 1234, "exp": [iat+ <5 mins],
  "aud": "trust-domain.example”
  "txn": [transaction identifier],
  "sub_id": { //… the subject identifier },
  "req_ctx": { //… requester context },
  "purp": "transaction purpose",
  "azd": { //… the authorization details }
}
```

Subject Identifiers in TraTs

```
{
  "sub_id": {
    "format": "email",
    "email": "atul@sgnl.ai"
  }
}
```

Defined in IETF “SecEvents Subject Identifiers” spec (to be RFC)

```
{
  "sub_id": {
    "format": "aliases",
    "aliases": [
      {
        "format": "email",
        "email": "atul@..."
      },
      {
        "format": "opaque",
        "id": "1234"
      }
    ]
  }
}
```

sgnl

Other Important Claims

- **Requester Context:** A claim that identifies the originating component (e.g. API gateway) and environment information such as IP address in the call chain
- **Purpose:** A claim that identifies the purpose of the original external call in the call chain

And...

sgnl

Authorization Details Claim

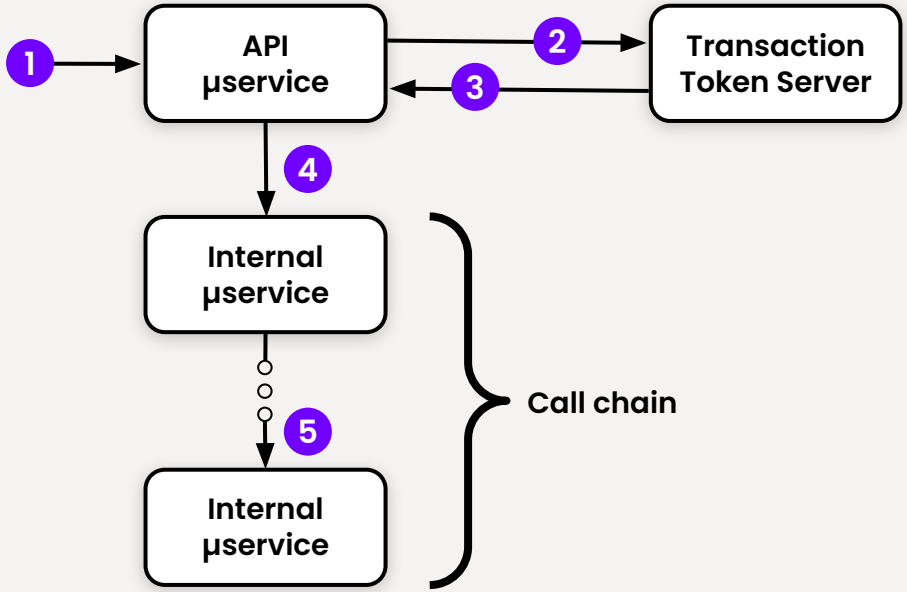
Authorization Details: Any information that needs to be preserved yet known throughout the call chain, e.g.

- The action initiated by the user
- The parameters of the action
- Any computed values about the transaction not in the original request that need to be preserved

```
{
    “azd”: {
        “action”: “BUY”,
// parameter of external call
        “ticker”: “MSFT”,
// parameter of external call
        “quantity”: “100”,
// parameter of external call
        “user_level”: “vip”
// computed value not present in
external call
    }
}
```

sgnl

Creating TraTs



1. User invokes external endpoint in API microservice
2. External microservice uses OAuth Token Exchange to get TraT from “TraT Server”
3. TraT Server verifies requesting service using SPIFFE and issues the TraT
4. API service uses the TraT to call internal service
5. Subsequent services in call chain use the TraT to invoke downstream services

sgnl

Requesting TraTs

A TraT Request is a OAuth 2.0 Token Exchange Request

- **`subject_token`** is external token
- **`subject_token_type`** is external token type
- **`rctx`** is an additional parameter in request that contains information required to generate the TraT

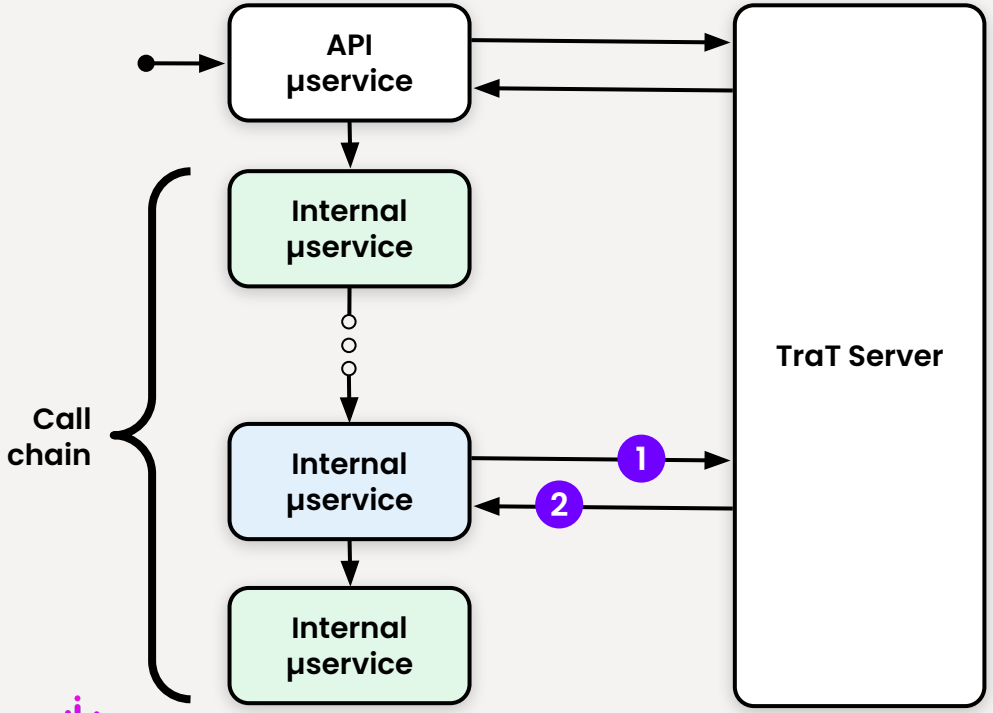
TraT Server Response to Creation Request

Transaction Token Response

- Issued **`token_type`** is **“`txn_token`”**
- Transaction Token is contained in **`access_token`** value
- Response MUST NOT contain:
  - **`expires_in`**, **`refresh_token`**, and **`scope`**

sgnl

Creating Replacement TraTs



1. User invokes external endpoint in API microservice
2. External microservice uses OAuth Token Exchange to get TraT from “TraT Server”

Service passing the TraT through

Service obtaining a replacement TraT

sgnl

Caution While Creating Replacement TraTs

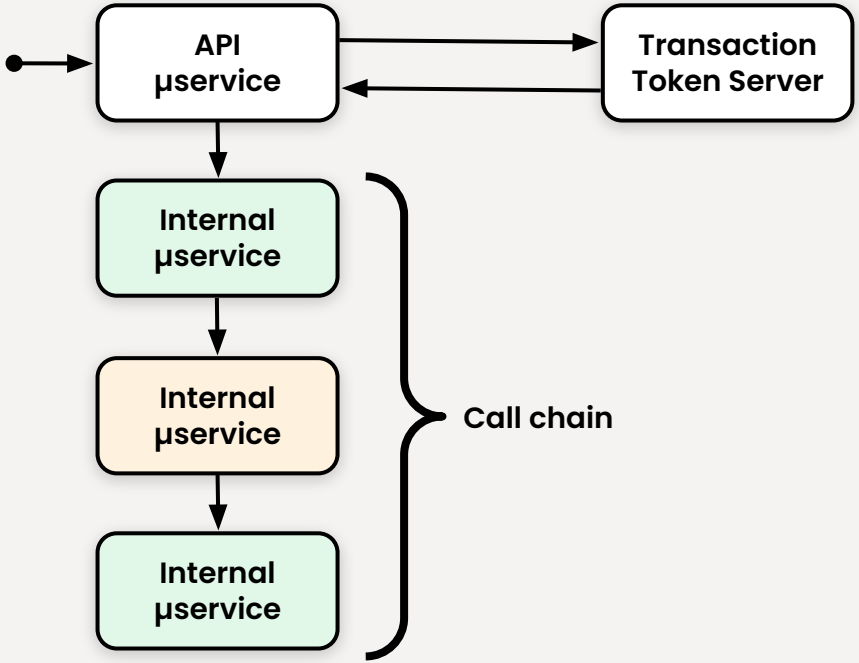
- Should not enable arbitrary modifications to previously asserted values
- May specialize the purpose
- May add to asserted values

Future Considerations

Call Chain Assurance through Nested TraTs

sgnl

Creating Nested Transaction Tokens



Service passing Transaction Token through

Service generating nested Transaction Token

sgnl

Nested Transaction Tokens

- Self-signed JWT Embedded Token, contains Transaction Token
  - May be nested recursively
- Benefits
  - Downstream service can verify that signing service was in the call chain
- Drawbacks
  - Token bloat
  - More trusted services

sgnl

Learn more about Continuous Access Management

TraTs spec GitHub repo

THANK YOU

@zirotrust tulshi

sgnl