clever cloud

Biscuit authorization tokens

Hi, I’m Geoffroy Couprie

- Rust developer at Clever Cloud
- sozu HTTP reverse proxy
- WebAssembly based FaaS platform
- Biscuit tokens
- nom parser combinators

- github: geal
- twitter: gcouprie

State of the art

State of the art

JWT
tokens signed by public key cryptography

Macaroons
offline attenuation

JSON Web Tokens

- mainly signed by public key cryptography
- (also priv key, encryption, etc)
- contains data (user ID, etc)
- used to store session information: the server can verify that the data was not tampered with
- used in OAuth and OIDC
- some pitfalls: alg=none vulnerability, revocation...

State of the art

JWT
tokens signed by public key cryptography

Macaroons
offline attenuation

Macaroons

- signed with private key crypto (HMAC)
- contains *caveats*: conditions over the request that must be verified
- offline attenuation
- pitfalls: caveat language not defined, needs the private key to verify

State of the art

JWT
tokens signed by public key cryptography

Macaroons
offline attenuation

Could we get macaroons with public key cryptography?

- separate macaroon creator from verifier
- transmit a macaroon from service to service: no need to share the key

Biscuit

Biscuit

Summary
a mix of JWT and macaroons

Biscuit

Block 0	pub root	can read and write /folder1/file1 can read /folder2/file1
Block 1	pub key1	restrict to read operations
Block 2	pub key2	restrict to path prefix /folder1/
signature = sign(block0, root) + sign(block1, key1) + sign(block 2, key2)		

Verifier:

- knows root public key
- file is /folder1/file1
- operation is read
- verify the checks
- verify that we have the right to read /folder1/file1
- success!

Applications

Applications

API authorization
more flexibility for clients

Microservices
bearer tokens with attenuation

API authorization

the promise of OAuth: delegated authorization

in reality: coarse grained authorization, token with too many rights, complex rights management interfaces

Example: how to reduce the rights of your Github token for CI?

Applications

API authorization
more flexibility for clients

Microservices
bearer tokens with attenuation

API authorization with Biscuit

let users attenuate their token

- shorter expiration time
- limit to a specific project or file
- limit origin IP…

as long a the server provides the facts, they can be used in authorization rules

Applications

API authorization
more flexibility for clients

Microservices
bearer tokens with attenuation

API authorization with Biscuit

Apache Pulsar example:

- we host a multi-tenant Pular cluster
- we give each customer a Biscuit token with full rights on their namespace
- they attenuate their token to get specific rights for each application

-> a token that can only read on topic A and write on topic B
-> a token that can read on topic C but only for the next hour

all the other rules we defined still apply, customers define their own on top

Applications

API authorization
more flexibility for clients

Microservices
bearer tokens with attenuation

Microservices authorization

How do you authorize requests between microservices?

each service has its own authorization:

- services must be connected manually to each other
- Confused deputy problem: authorization is tied to the service, not the request

Applications

API authorization
more flexibility for clients

Microservices
bearer tokens with attenuation

Microservices authorization

How do you authorize requests between microservices?

centralized authorization:

- either through the API gateway, or a central authorization service
- single point of failure
- great overhead

Applications

API authorization
more flexibility for clients

Microservices
bearer tokens with attenuation

Microservices authorization

How do you authorize requests between microservices?

decentralized with bearer tokens (JWT):

- the same token with full rights is used everywhere
- a service could keep an old token and reuse it

Applications

API authorization
more flexibility for clients

Microservices
bearer tokens with attenuation

Microservices authorization with Biscuit

bearer tokens get attenuated before transmission to the next service

- from a full rights token, get a short lived token
- limit rights when requesting the next service:
  - ex: give rights to look up inventory, but not invoicing

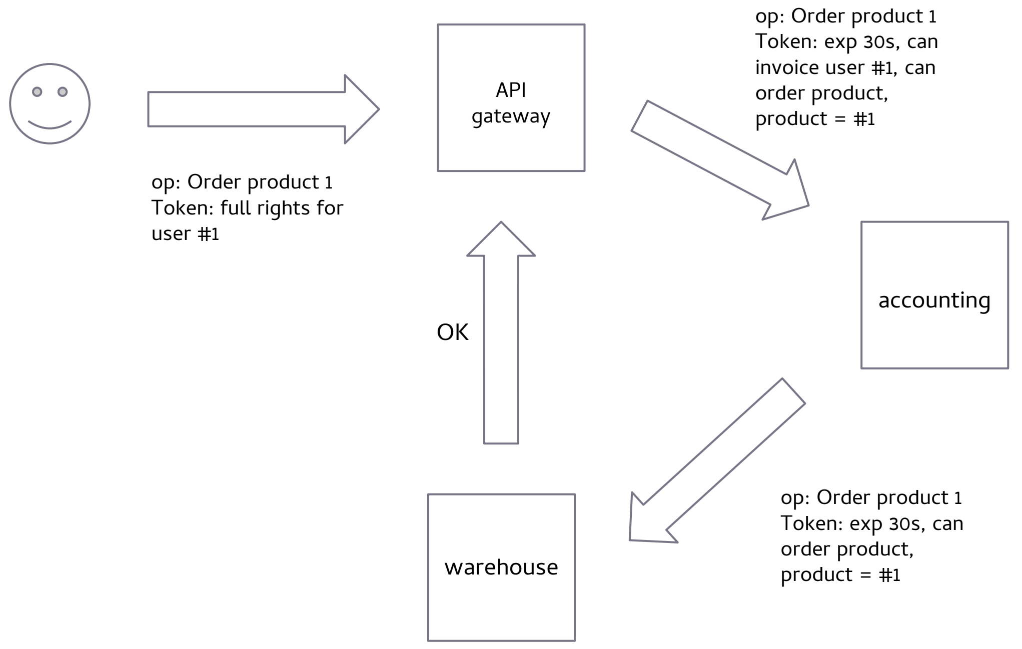
services will only act with a very limited token

Applications

API authorization
more flexibility for clients

Microservices
bearer tokens with attenuation

Microservices authorization with Biscuit


API gateway
op: Order product 1
Token: exp 30s, can invoice user #1, can order product, product = #1
op: Order product 1
Token: full rights for user #1
accounting
OK
warehouse
op: Order product 1
Token: exp 30s, can order product, product = #1

Technical details

Technical details

Summary
a mix of JWT and macaroons

Cryptography
signature aggregation

Serialization
Protobuf

Biscuit

- public key cryptography (aggregated signatures)
- offline attenuation
- authorization language based on Datalog
- can contain data, code and authorization checks
- specifies revocation for a token and all derived tokens
- can extract data for audit and replay

Technical details

Summary
a mix of JWT and macaroons

Cryptography
signature aggregation

Serialization
Protobuf

Cryptography

Signature aggregation: sign separately multiple messages, then assemble them in one signature

- based on *aggregated gamma signatures*(https://eprint.iacr.org/2018/414)
- implemented with *Ristretto* (https://ristretto.group/)
- can be implemented on libsodium (example code available)

Technical details

Summary
a mix of JWT and macaroons

Cryptography
signature aggregation

Serialization
Protobuf

Serialization

- a token contains a list of blocks
- each block is a protobuf structure containing data and authorization rules
- each block is signed
- attenuation is done by adding a block and aggregating its signature with the token’s

Technical details

Summary
a mix of JWT and macaroons

Cryptography
signature aggregation

Serialization
Protobuf

Serialization

```
message Biscuit {
 required bytes authority = 1;
 repeated bytes blocks = 2;
 repeated bytes keys = 3;
 required Signature signature = 4;
}
message Block {
 required uint32 index = 1;
 repeated string symbols = 2;
 repeated Fact facts = 3;
 repeated Rule rules = 4;
 repeated Check checks = 5;
 optional string context = 6;
 optional uint32 version = 7;
}
```

Datalog

Facts

a Datalog *fact* is data:

```
parent("Alice", "Bob");
parent("Bob", "Charles");
parent("Charles", "Denise");
```

can be seen as:

parent		
	Alice	Bob
	Bob	Charles
	Charles	Denise

Datalog

- Facts
- Rules
- Checks
- Allow/deny policies

Rules

a *rule* is used to query data:

```
parent_of_charles($name) <-
  parent($name, "Charles");
```

it can be translated to SQL:

```
SELECT DISTINCT name from parent where child = "Charles";
```

Result: `parent_of_charles(“Bob”)`

Datalog

Rules

a *rule* can generate new *facts*

```
 grandparent($grandparent, $child) <-
   parent($grandparent, $parent),
   parent($parent, $child);
```

could be seen as:

```
create the fact grandparent($grandparent, $child)
  IF
    there is a fact parent($grandparent, $parent)
    AND there is a fact parent($parent, $child)
    with matching $parent variable
```

SQL version:

```
INSERT INTO grandparent( name, grandchild )
  SELECT A.name as name, B.child as grandchild
  FROM parent A, parent B

  WHERE A.child = B.name;
```

Datalog

- **Facts**
- **Rules**
- **Checks**
- **Allow/deny policies**

Rules

a *rule* can generate new *facts*

```
grandparent($grandparent, $child) <-
  parent($grandparent, $parent),
  parent($parent, $child);
```

Creates:

```
grandparent("Alice", "Charles");
grandparent("Bob", "Denise");
```

parent		
	Alice	Bob
	Bob	Charles
	Charles	Denise

grandparent		
	Alice	Charles
	Bob	Denise

Checks

a *check* is a condition over the request

- all checks must pass
- they can be provided by the token or the verifier

```
check if operation(#ambient, #read);

check if
   time(#ambient, $date),
   $date <= 2018-12-20T00:00:00+00:00;
```

Allow/deny policies

allow and *deny* policies are tested one by one until one matches

```
allow if
  operation(#ambient, $op),
  resource(#ambient, $res),
  right(#authority, $res, $op);

deny if true;
```

Datalog

Facts

Rules

Checks

Allow/deny policies

Example: RBAC

the token would contain `user(#authority, #user_123)`
On the verifier’s side:

```
role(#authority, #user_123, "team1”, #member);
role(#authority, #user_123, "team2”, #manager);
rights(#authority, "team1”, #member, "PROJECT1", [#read]);
rights(#authority, “team1”, #manager, "PROJECT1", [#read, #write, #delete]);

// a manager automatically gets the right of a member
role(#authority, $user_id, $team1, #member) <-
    role(#authority, $user_id, $team, #manager);

allow if
  resource(#ambient, $project),
  operation(#ambient, $op),
  user(#authority, $user_id),
  role(#authority, $user_id, $team, $role),
  rights(#authority, $team, $role, $project, $rights),
  $rights.contains($op);

// this catch-all policy will refuse the request
deny if true
```

Project status

Project status

Implementations

Real world usage

Links

Implementations

- Rust (with C and Webassembly bindings)
- Java
- Go
- Haskell

In preparation:

- C#
- Swift
- who’s next?...

Real world usage

- Biscuit Pulsar
- a (stealth) startup using a Biscuit token as license
- (not released yet) a layer for FoundationDB using Biscuit to specify which key prefixes are accessible

Do you have fun ideas and applications? Come talk to me!

Project status

Implementations

Real world usage

Links

Links

- Specification https://github.com/clevercloud/biscuit
- Playground https://play-with-biscuit.cleverapps.io/
- implementations
  - https://github.com/clevercloud/biscuit-rust
  - https://github.com/clevercloud/biscuit-java
  - https://github.com/biscuit-auth/biscuit-go
- articles
  - intro to Biscuit https://www.clever-cloud.com/blog/engineering/2021/04/12/introduction-to-biscuit/
  - tutorial https://www.clever-cloud.com/blog/engineering/2021/04/15/biscuit-tutorial/

Clever Cloud Paris
137 rue vieille du temple 75003 Paris

Clever Cloud Nantes
3 rue de l’allier 44000 Nantes
02 85 52 07 69

https://www.clever-cloud.com

CONTACT

mail@clever-cloud.com

+33 2 85 52 07 69