Trusted Computing Base In Action

Machine to Machine API Calling

Bahram Jahanshahi
Senior Software Engineer at Agile Search
January 2022

Agenda

Part I

1. Security by Design
2. Secure Data in Transit
3. Credential
4. JWT Best Practices
5. Architecture

Part II

1. Resilience Engineering
2. DevSecOps
3. Consumer Driven Contract Testing

Agenda

Part I

1. **Security by Design**
2. Secure Data in Transit
3. Credential
4. JWT Best Practices
5. Architecture

Part II

1. Resilience Engineering
2. DevSecOps
3. Consumer Driven Contract Testing

Secure by Design

Domain Driven Design and Value Object

Use Strong Types

Common Security Vulnerabilities

Use memory-safe language

Simplicity Leads to Secure and Reliable Code

Domain Driven Design

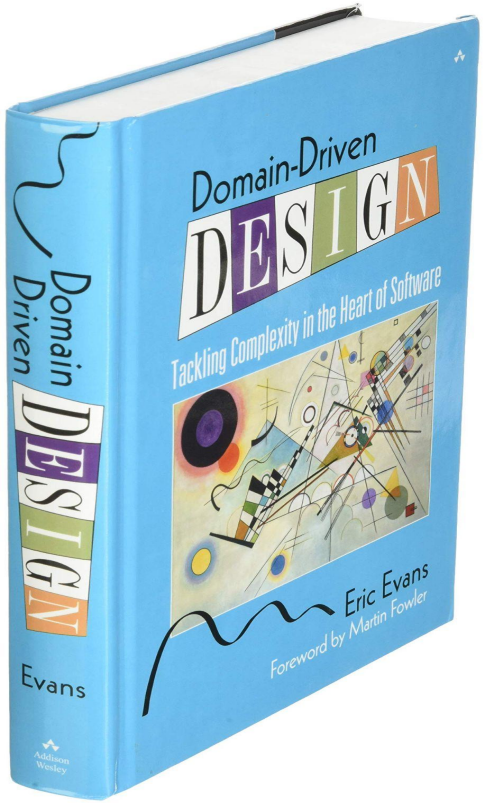
Bounded Context

Entities

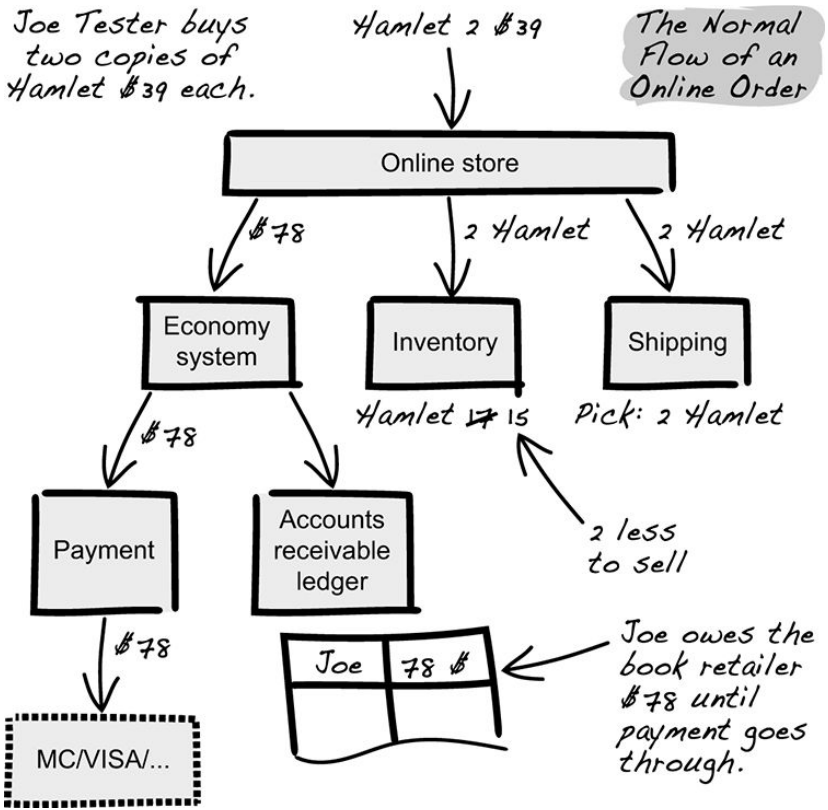
Value Objects

Aggregate

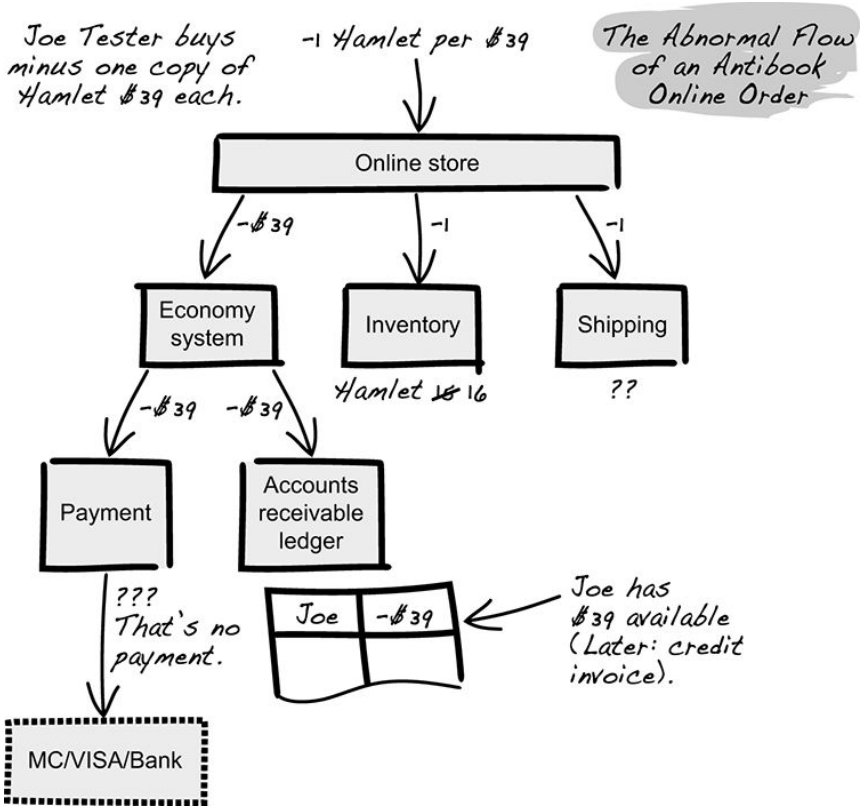
…



Anti-Hamlet


Joe Tester buys two copies of Hamlet $39 each.
Hamlet 2 $39
The Normal Flow of an Online Order
Online store
$78
2 Hamlet
2 Hamlet
Economy system
Inventory
Shipping
Hamlet ~~17~~ 15
Pick: 2 Hamlet
$78
Payment
Accounts receivable ledger
2 less to sell
$78
Joe
78 $
Joe owes the book retailer $78 until payment goes through.
MC/VISA/...

Anti-Hamlet: Buy minus one copy of book



DDD: Value Object

Never use an Integer as a quantity of books in the Order entity.

Instead of an Integer use a value object like OrderQuantity:

```
class OrderQuantity {
	Private final int value;
	public OrderQuantity(int value) {
		shouldNotLessThanZero(value);
		shouldNotMoreThan240(value);
		setValue(value)
	}
}
```

Use Strong Types

AddUserToGroup(string, string)

It’s unclear whether the group name is provided as the first or the second argument.

Rectangle(3.14, 5.67)

What is the order of height and width?

Circle(double)

It expect radius or diameter?

Use Strong Types

Add(User(“alice”), Group(“root-user”)

Rectangle(Width(3.14), Height(5.67))

Circle(Radius(1.23))

User, Group, Width, Height, and Radius are strong type wrappers around string and double primitives.

Common Security Vulnerabilities

SQL Injection Vulnerability: TrustedSqlString

Preventing XSS: SafeHtml

Simplicity Leads to Secure and Reliable Code

Avoid Multi-Level Nesting

Multi-level nesting is common anti-pattern that can lead to simple mistakes.

```
function register()
{
    if (!empty($_POST)) {
        $msg = '';
        if ($_POST['user_name']) {
            if ($_POST['user_password_new']) {
                if ($_POST['user_password_new'] === $_POST['user_password_repeat']) {
                    if (strlen($_POST['user_password_new']) > 5) {
                        if (strlen($_POST['user_name']) < 65 && strlen($_POST['user_name']) > 1) {
                            if (preg_match('/^[a-z\d]{2,64}$/i', $_POST['user_name'])) {
                                $user = read_user($_POST['user_name']);
                                if (!isset($user['user_name'])) {
                                    if ($_POST['user_email']) {
                                        if (strlen($_POST['user_email']) < 65) {
                                            if (filter_var($_POST['user_email'], FILTER_VALIDATE_EMAIL)) {
                                                create_user();
                                                $_SESSION['msg'] = 'You are now registered so please login';
                                                header('Location: ' . $_SERVER['PHP_SELF']);
                                                exit();
                                            } else $msg = 'You must provide a valid email address';
                                        } else $msg = 'Email must be less than 64 characters';
                                    } else $msg = 'Email cannot be empty';
                                } else $msg = 'Username already exists';
                            } else $msg = 'Username must be only a-z, A-Z, 0-9';
                        } else $msg = 'Username must be between 2 and 64 characters';
                    } else $msg = 'Password must be at least 6 characters';
                } else $msg = 'Passwords do not match';
            } else $msg = 'Empty Password';
        } else $msg = 'Empty Username';
        $_SESSION['msg'] = $msg;
    }
    return register_form();
}
```

Simplicity Leads to Secure and Reliable Code

Eliminate YAGNI Smells

You **A**ren’t **G**onna **N**eed **I**t

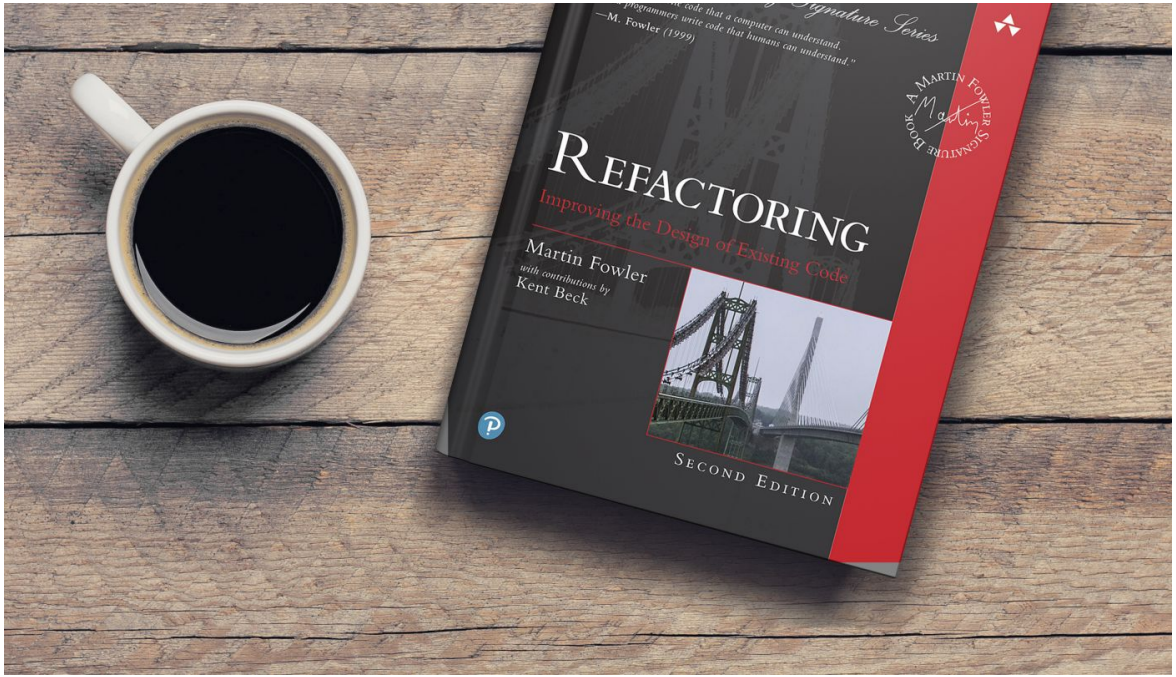
- This principle recommends implementing only the code that you need.
- YAGNI code adds unnecessary complexity, because it needs to be documented, tested, and maintained.
- Avoiding YAGNI code leads to improve reliability, and simpler code leads to fewer security bugs, fewer opportunities to make mistakes, and less development time spent maintaining unused code.

Simplicity Leads to Secure and Reliable Code

Repay Technical Debt (TODO, FIXME)

Simplicity Leads to Secure and Reliable Code

Refactoring



Simplicity Leads to Secure and Reliable Code

Refactoring Golden Rule

Never mix refactoring and functional changes

in a single commit to the code repository.

It’s really hard to review!

Agenda

Part I

1. Security by Design
2. **Secure Data in Transit**
3. Credential
4. JWT Best Practices
5. Architecture

Part II

1. Resilience Engineering
2. DevSecOps
3. Consumer Driven Contract Testing

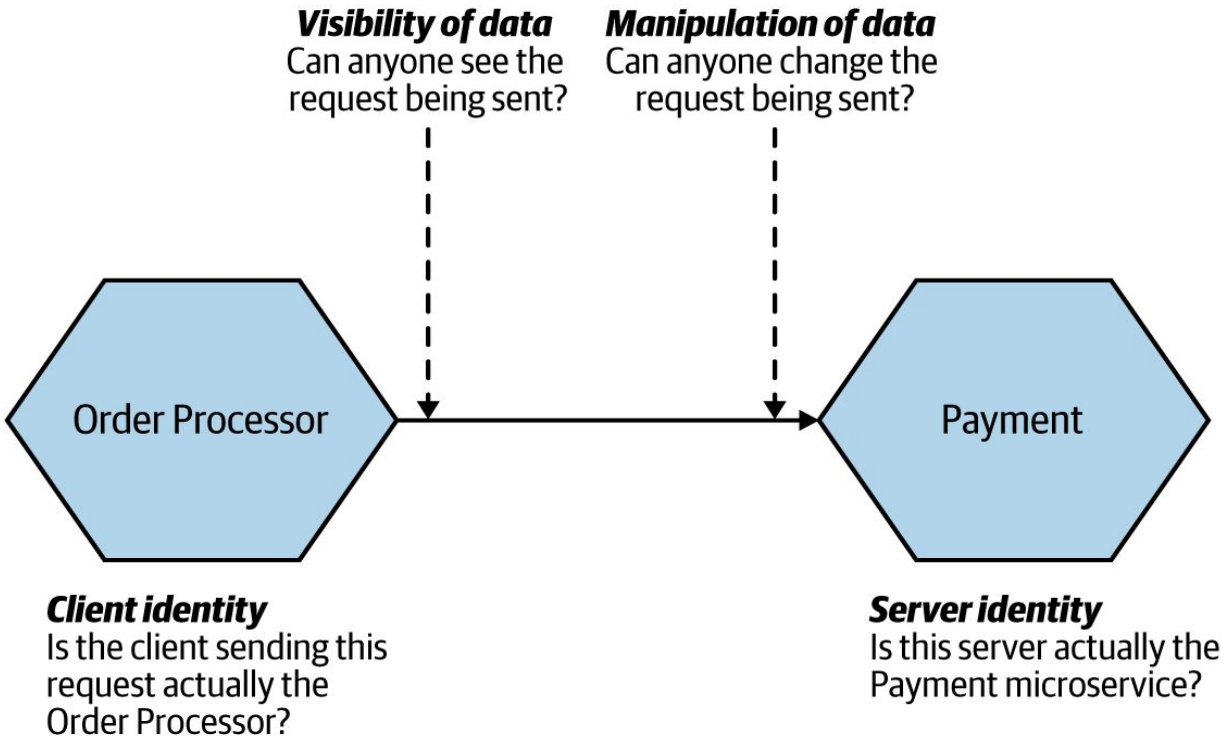
Securing Data in Transit

Secure Data in Transit

The four main concerns when it comes to data in transit

1. Server Identity
2. Client Identity
3. Visibility of Data
4. Manipulation of Data

Secure Data in Transit



Secure Data in Transit

HTTPS Everywhere

Mutual TLS

HMAC (hash-based message authentication code)

Agenda

Part I

1. Security by Design
2. Secure Data in Transit
3. **Credentials**
4. JWT Best Practices
5. Architecture

Part II

1. Resilience Engineering
2. DevSecOps
3. Consumer Driven Contract Testing

Credentials

Credentials give a person (or computer) access to some form of restricted resource.

Two categories of credentials:

1. User credentials

   Such as username and password. ***Often the weakest point of our system***

2. Secrets

   Pieces of information that are critical to running the services.

Credentials: User Credential

Some advices:

1. Embrace Password Manager
2. Use Long Password
3. …

Recommend read this article of Troy Hunt in detail:

https://www.troyhunt.com/passwords-evolved-authentication-guidance-for-the-modern-era/

Credentials: Secret

Secrets might be:

1. Certificates for TLS
2. SSH keys
3. Public/Private API key pairs
4. Credentials for accessing database

Considerations:

1. **Creation** (How to create secret in the first place?)
2. **Distribution** (How to make sure it gets to the right place?)
3. **Storage** (Only authorized parties can access?)
4. **Monitoring** (Do we know how the secret is being used?)
5. **Rotation** (Are we able to change the secret without causing problem?)

Credentials: Secret Tools

Tools to manage all of the considerations:

1. Kubernetes built-in solution
2. Hashicorp’s Vault
3. AWS Secret Manager

Credentials: Issues

Across both user credentials and secrets, we have to consider these issues:

1. Rotation

   Ideally, we want to rotate credentials frequently to limit the damage someone can do if they gain access to credentials. Hashicorp’s Vault can generate time-limited credentials and the credentials can be generated on the fly.

2. Revocation
3. Least Privilege Principle

Agenda

Part I

1. Security by Design
2. Secure Data in Transit
3. Credential
4. **JWT Best Practices**
5. Architecture

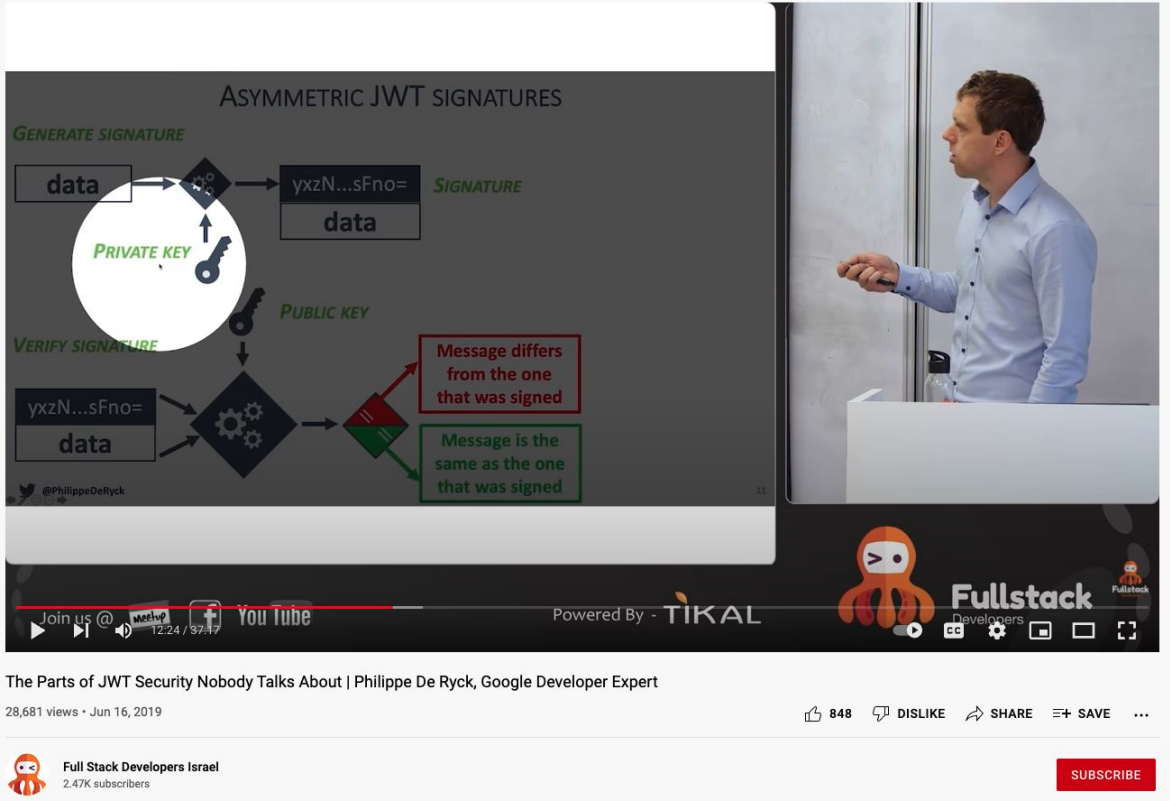
Part II

1. Resilience Engineering
2. DevSecOps
3. Consumer Driven Contract Testing

JWT Best Practice

The Part of JWT Security
Nobody Talks About

https://www.youtube.com/watch?v=DPrhem174Ws



JWT Best Practice

Encoded PASTE A TOKEN HERE


```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ
zdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IlBoaWx
pcHBlIERlIFJ5Y2siLCJyb2xlcyI6InVzZXIgcmV
zdGF1cmFudG93bmVyIiwiaWF0IjoxNTE2MjM5MDI
yfQ.KPjhyE9oi83uehgw6Lm_0yAZzRuJhcUqXETD
2AIrF2A
```


Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE


```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

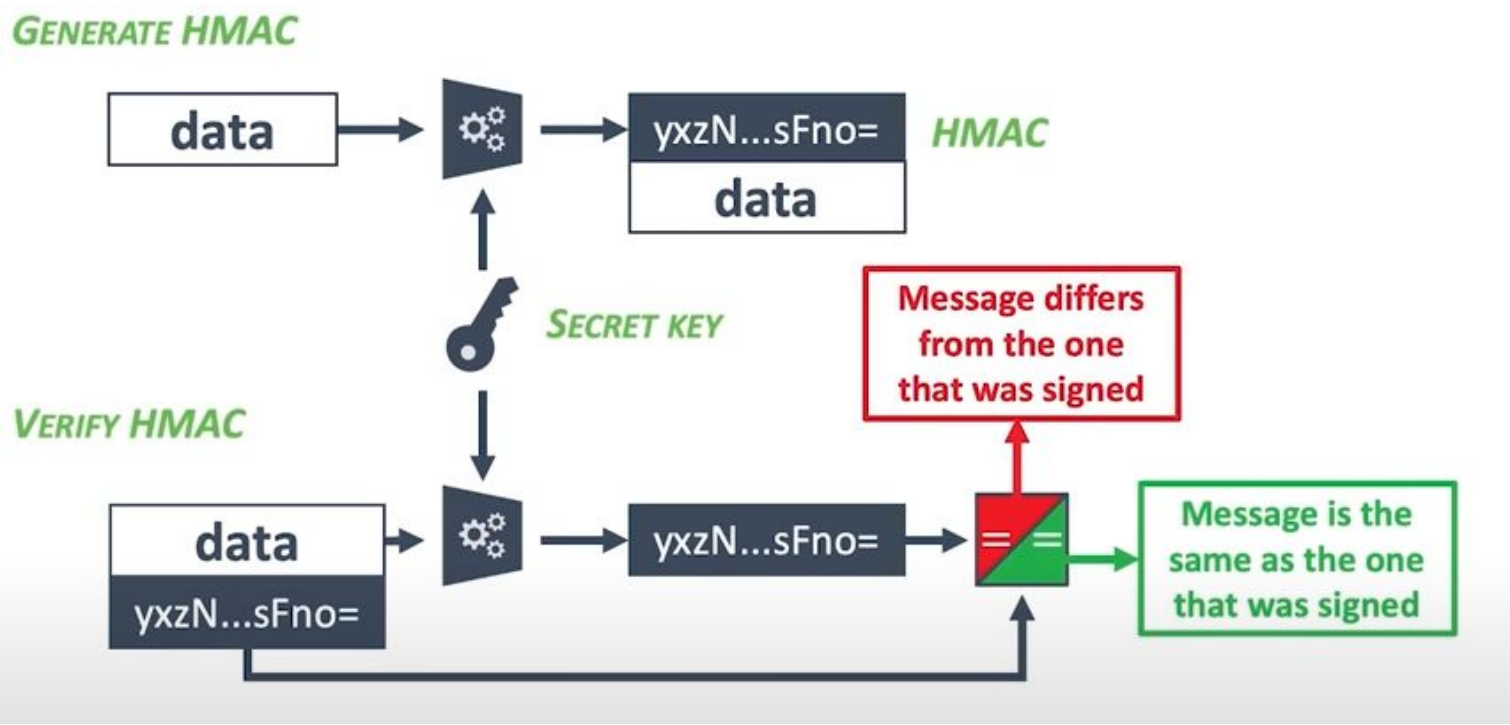

PAYLOAD: DATA


```
{
  "sub": "1234567890",
  "name": "Philippe De Ryck",
  "roles": "user restaurantowner",
  "iat": 1516239022
}
```


VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  SuperSecretHMACKey
) secret base64 encoded
```

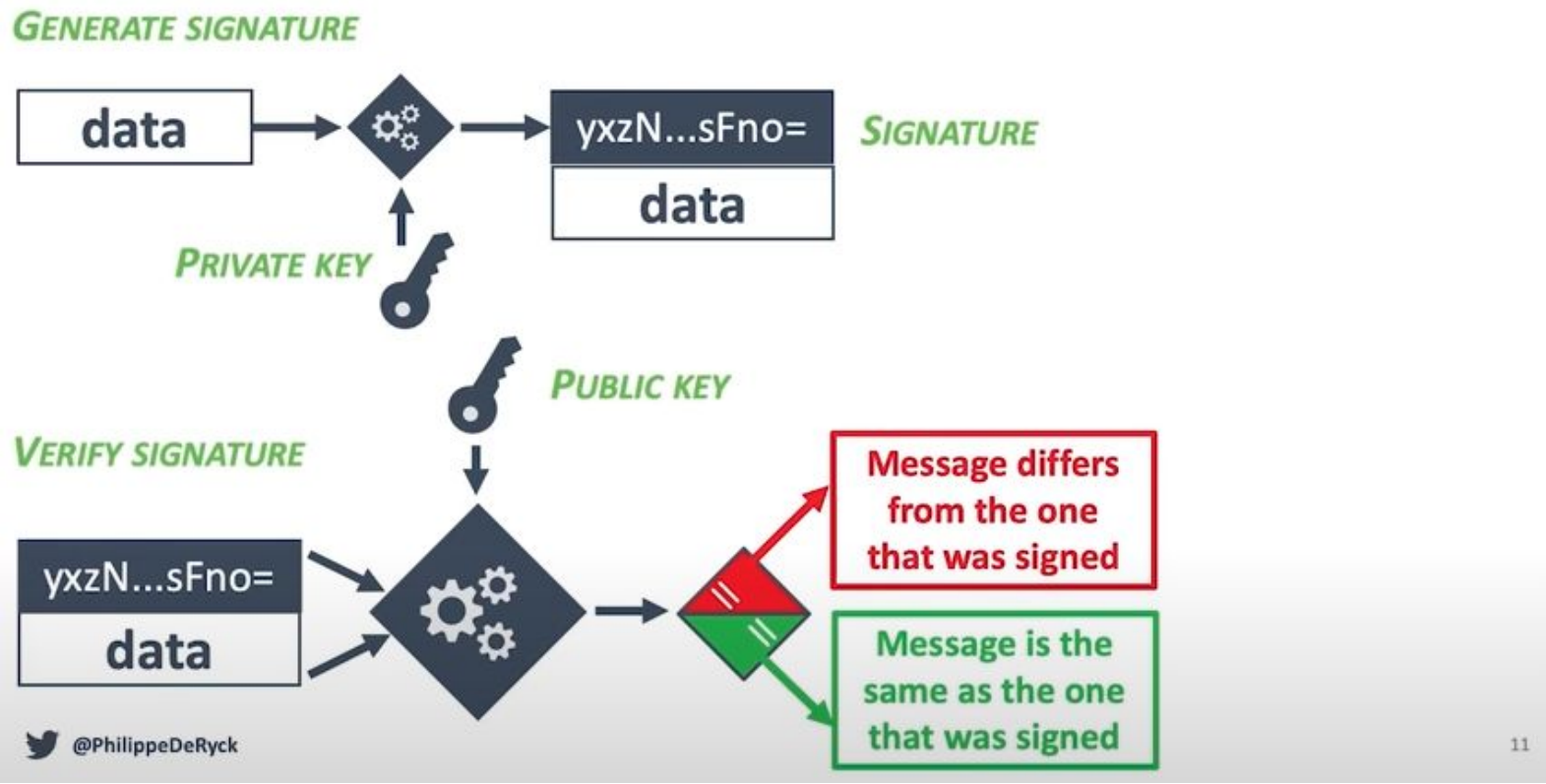
Symmetric JWT Signature



Never Share Your Secrets

Cracking a JWT sign with weak keys is possible via brute force attack

ASymmetric JWT Signature



How Would You Solve the

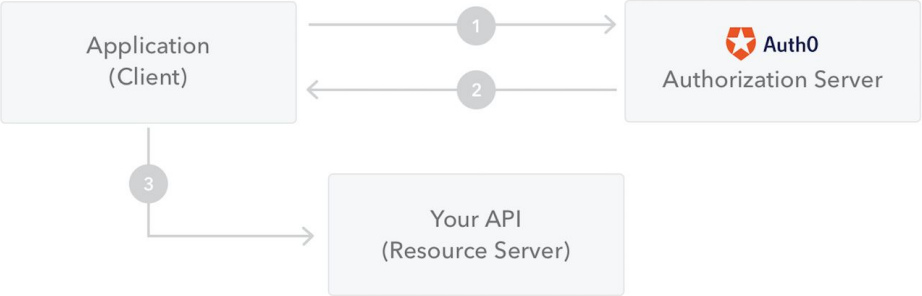
KEY MANAGEMENT

Problem with JWS?

Can We Skip to the Good Part?

Auth0 M2M

1. Client requests for token
2. Auth0 server provides a JWT token
3. Client send a request to the Server with the token in the header



Server knows how to verify token.
It has the public key which is provided by Auth0.

Agenda

Part I

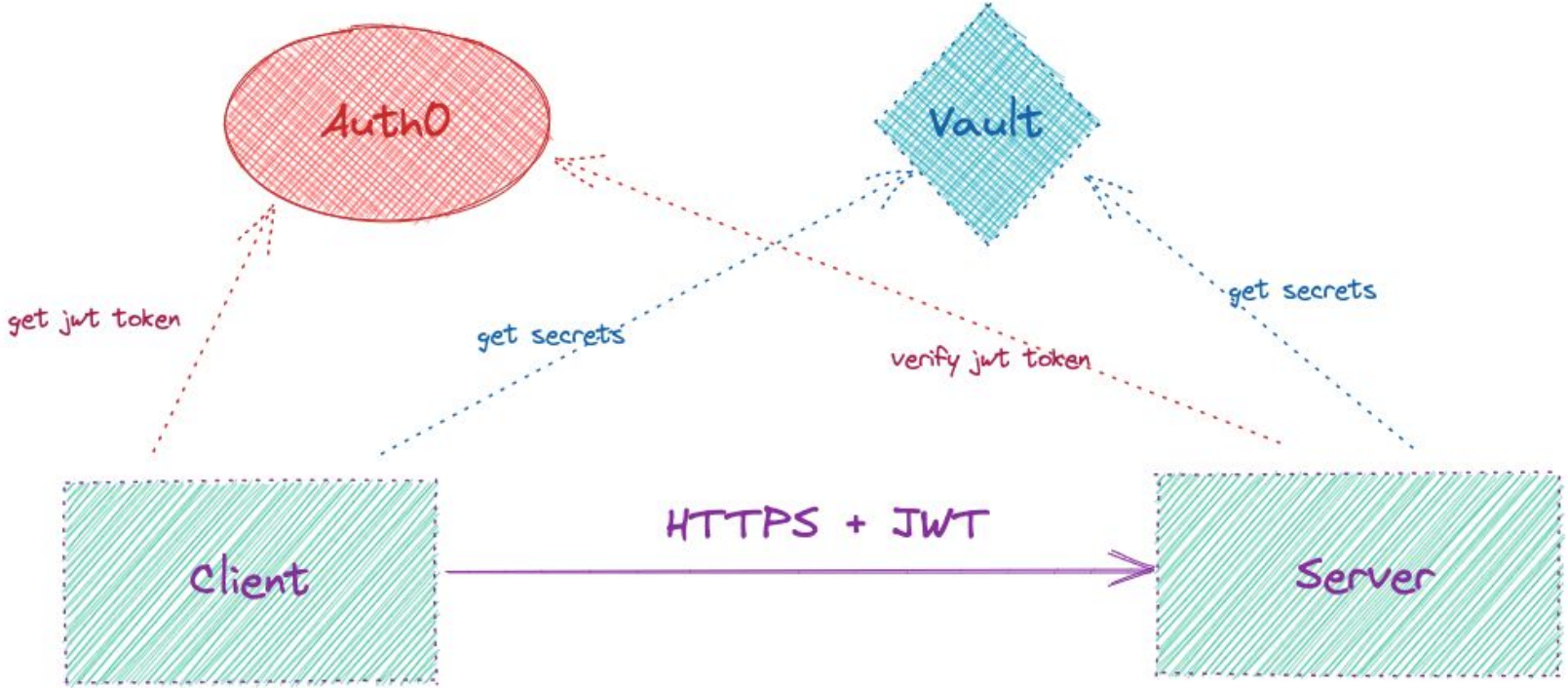
1. Security by Design
2. Secure Data in Transit
3. Credential
4. JWT Best Practices
5. **Architecture**

Part II

1. Resilience Engineering
2. DevSecOps
3. Consumer Driven Contract Testing

Let’s Design

Architecture



Framework for RPC Backend

Most RPC backends follow a similar structure:

1. Logging
2. Authentication
3. Authorization
4. Rate Limiting (Throttling)

Predefined Interceptors

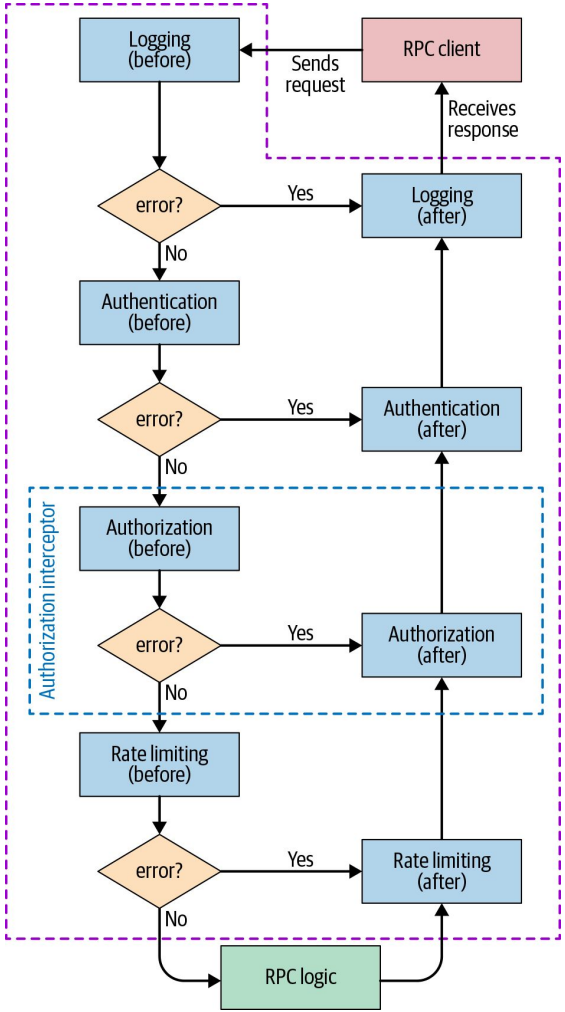
Instead of reimplementing this functionality for every single RPC, it’s recommended using a framework that can hide the implementation details of these building Block.

Predefined Interceptors

Interceptors share state through a ***context object*** that they pass to each other.

For example:
The authentication interceptor (before state) can handle all the cryptographic operations.

Execution time:
By using *context object* to track request execution time



Part II

1. Resilience Engineering
2. DevSecOps
3. Consumer Driven Contract Testing

Part II

1. **Resilience Engineering**
2. DevSecOps
3. Consumer Driven Contract Testing

Build With Failure In Mind

Build With Failure In Mind

Build With Failure In Mind

Fly Two Mistakes High

Fly Two Mistakes High

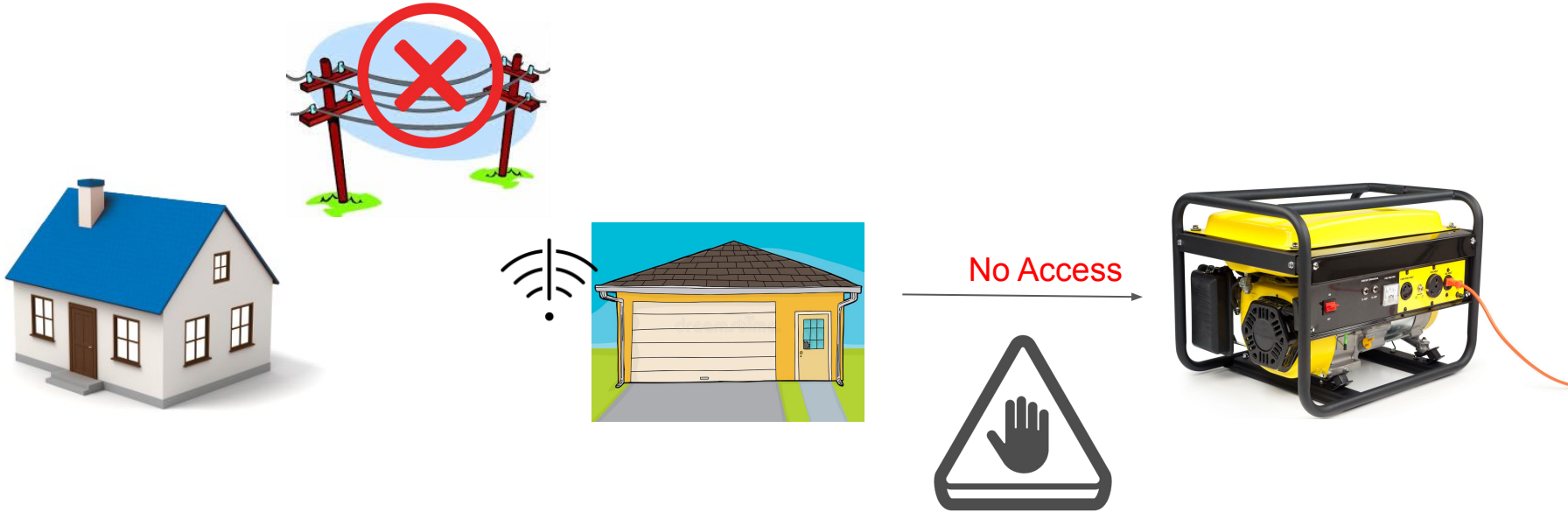
Fly Two Mistakes High

What happens if the database backups are stored in the same datacenter?

Fly Two Mistakes High

access

Fly Two Mistakes High



Chaotic Environment

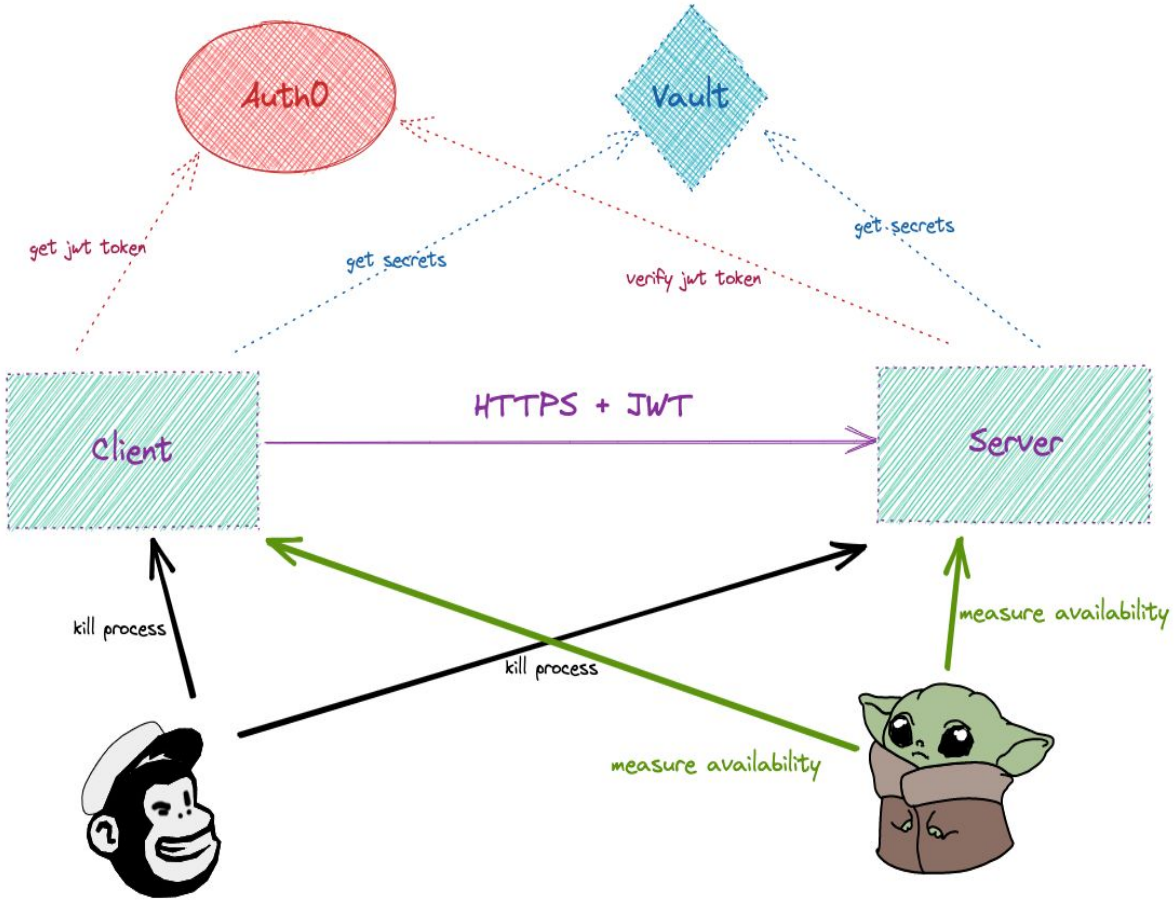
Chaotic Environment: Reality Of Our World

vs

Chaotic Environment: Reality Of Our World

Are you ready for failure?

Chaotic Environment

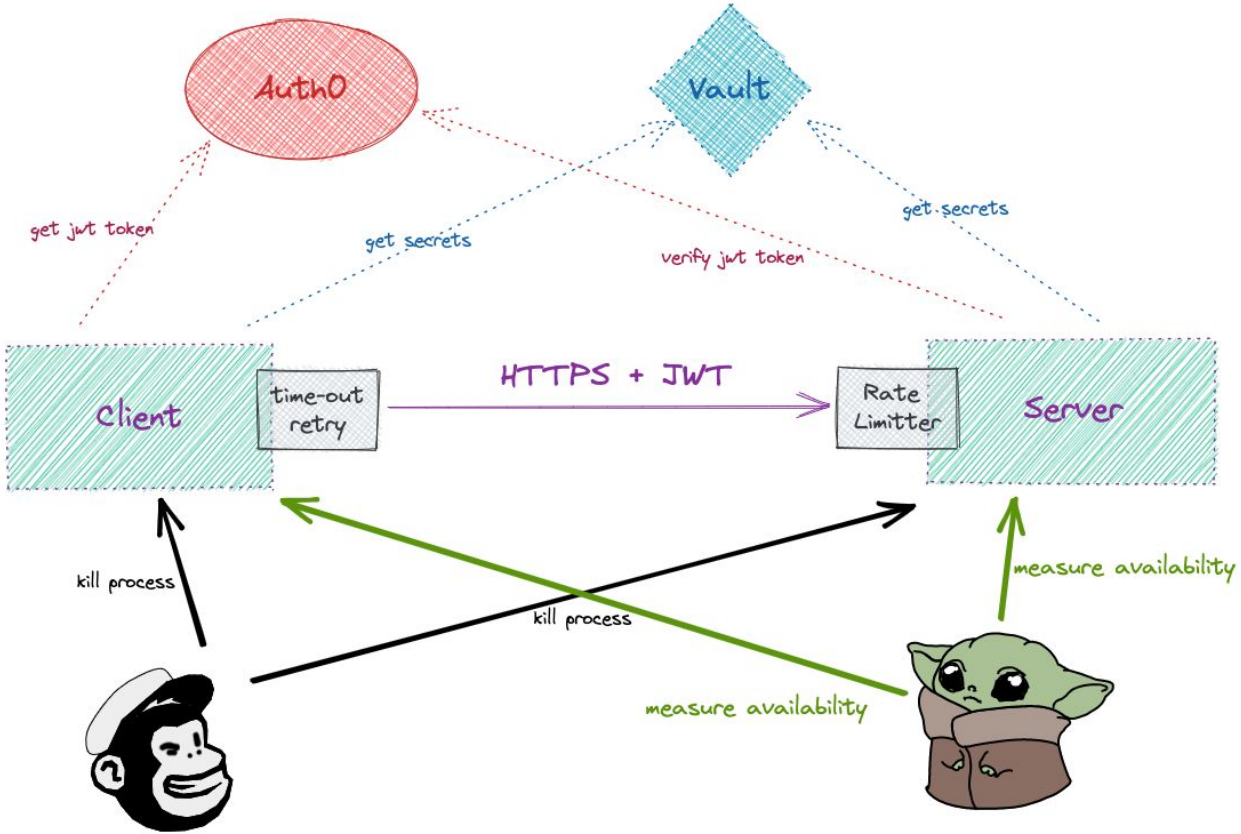


Resilience Engineering

What happens if downstream is not available?

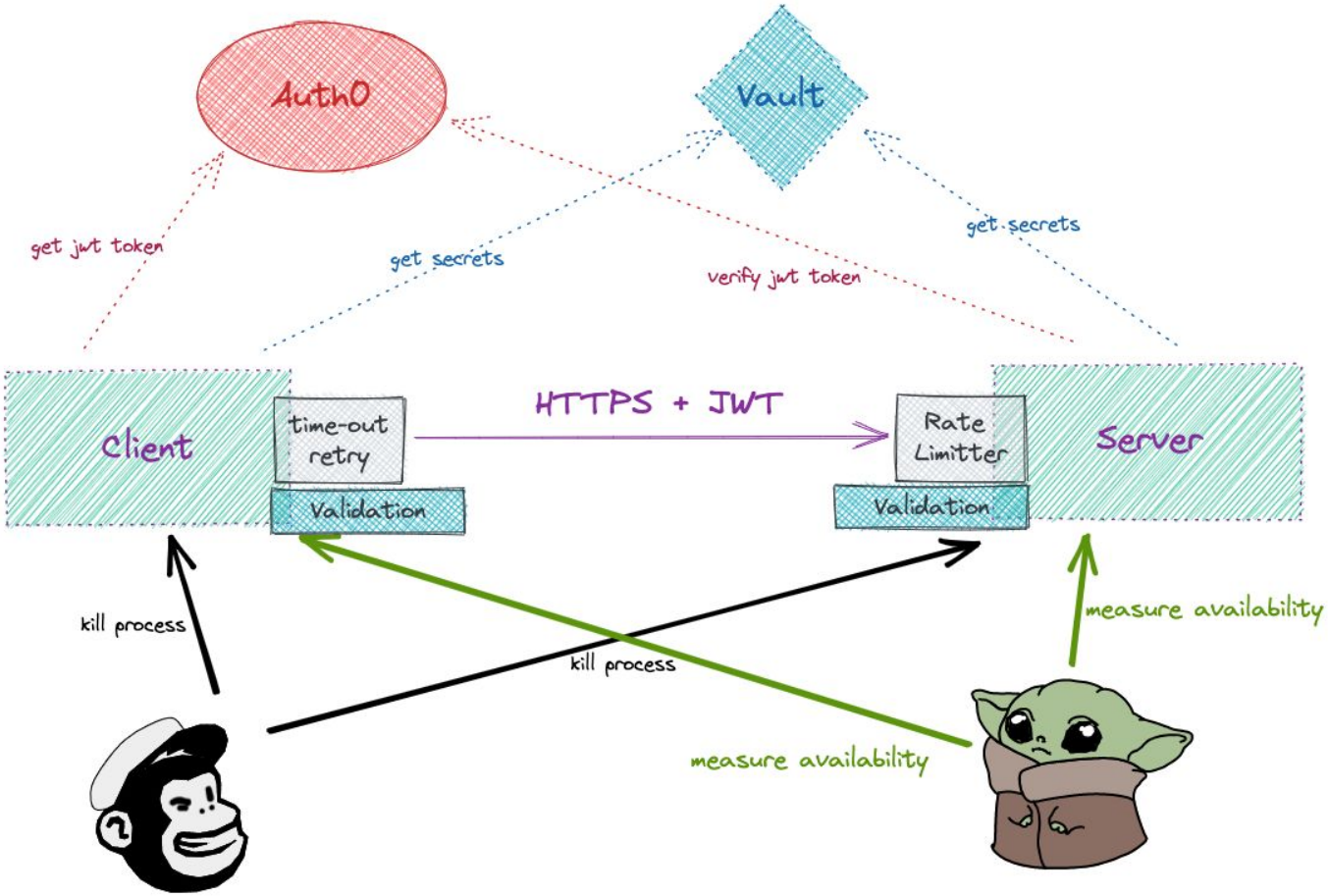
(Time-out, Retry, Rate Limiting, Circuit Breaker)

Time-out,
Retry,
and Rate Limiter



Validation & Correctness

Validation & Correctness



Functionality Degradation

Part II

1. Resilience Engineering
2. **DevSecOps**
3. Consumer Driven Contract Testing

DevSecOps

Test Security Vulnerabilities in a Automation Process

Check Container Security Issues

Defence In Depth

Part II

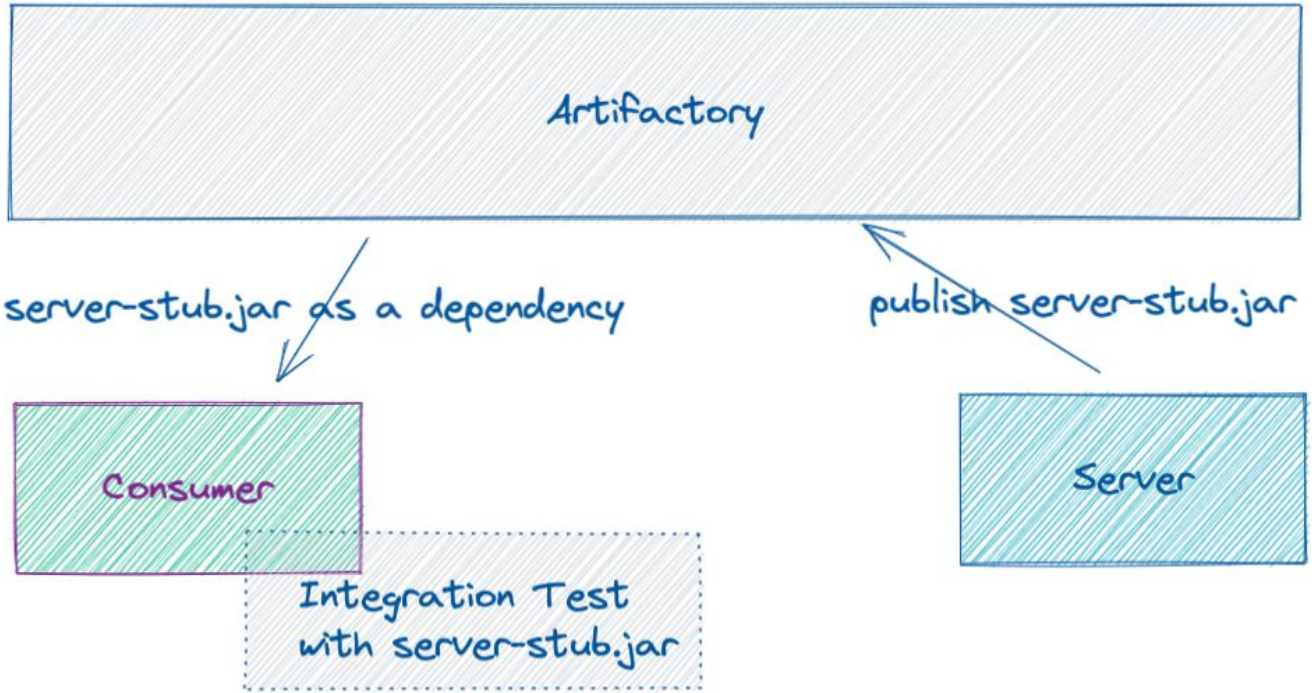
1. Resilience Engineering
2. DevSecOps
3. **Consumer Driven Contract Testing**

Consumer Driven Contract Testing

Helps Providers make changes without being scared of accidentally breaking their consumers

Contract testing lets everyone relax and be assured that the APIs won’t up and die.

Consumer Driven Contract Testing



Consumer Driven Contract Testing

Swagger Codegen

Swagger Codegen can simplify your build process by generating server stubs and client SDKs for any API, defined with the OpenAPI (formerly known as Swagger) specification, so your team can focus better on your API's implementation and adoption.

Spring Cloud Contract

Spring Cloud Contract is a project that, simply put, helps us write Consumer-Driven Contracts (CDC). This ensures the contract between a Producer and a Consumer, in a distributed system – for both HTTP-based and message-based interactions.

Further Works

Chaos Engineering

Observability and Telemetry

Scalability and Availability