USING RACK::ATTACK TO THROTTLE MALICIOUS REQUESTS

THE BARBICAN

Ben Carle

FullStack Labs

CASE STUDY

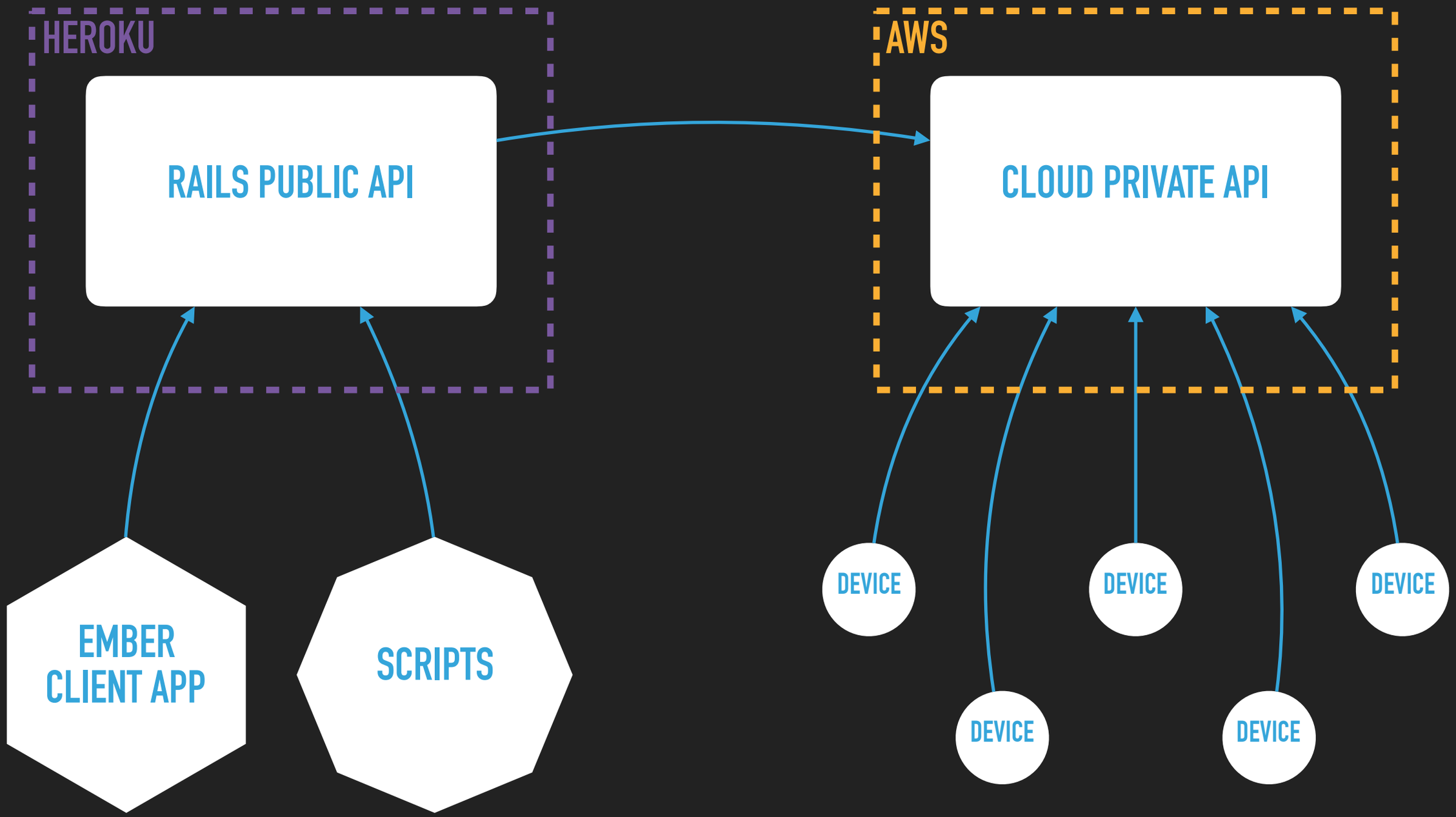
- The Setup
- The Problem
- The Improvement
- The New Reality

THE SETUP

MEET OUR APPLICATION

- Rails API
  - User Authentication
  - CRUD on
    - Locations
    - Devices

THE ARCHITECTURE



THE PROBLEM

NORMAL REQUESTS

- GET /assets/application-0dc8eafb85e36c7009fccbb6e6ef96ef.css
- GET /assets/application-5d5094458c36ec6a9903f157d746c770.js
- GET /assets/vendor/navbar-logo.png

NOT-SO-NORMAL REQUESTS

- `GET /assets/vendor/../../../../../../../../etc/passwd%C0%80.jsp`
- `GET /assets/vendor/../../../../../../../../windows/win.ini%C0%80.jsp`
- `GET /assets/../../../WEB-INF/web.xml%C0%80.jsp`
- `GET /assets/%C0%AE%C0%AE/%C0%AE%C0%AE/%C0%AE%C0%AE/WEB-INF/web.xml`
- `GET /assets/vendor/%C0%AE%C0%AE/%C0%AE%C0%AE/%C0%AE%C0%AE/%C0%AE%C0%AE/%C0%AE%C0%AE/%C0%AE%C0%AE/%C0%AE%C0%AE/%C0%AE%C0%AE/%C0%AE%C0%AE/etc/passwd`
- `GET /assets/%C0%AE%C0%AE\%C0%AE%C0%AE\%C0%AE%C0%AE\%C0%AE%C0%AE\%C0%AE%C0%AE\%C0%AE%C0%AE\%C0%AE%C0%AE\%C0%AE%C0%AE\%C0%AE%C0%AE\%C0%AE%C0%AE\%C0%AE%C0%AE\%C0%AE%C0%AE\%C0%AE%C0%AE\%C0%AE%C0%AE\%C0%AE%C0%AE\%C0%AE%C0%AE\etc/passwd`

INJECTION ATTACK

- Send requests designed to
  - Directly access sensitive data
  - Execute code in server app environment
    - Rails
  - Execute code outside app environment
    - BASH, Perl, etc.

INJECTION ATTACK

- Send requests designed to
  - Execute code in database environment
    - SQL
- Store code in DB for future execution
  - JavaScript

BUT WE ARE SAFE, RIGHT?

- We followed best practices
  - Secure server environment
  - Heroku

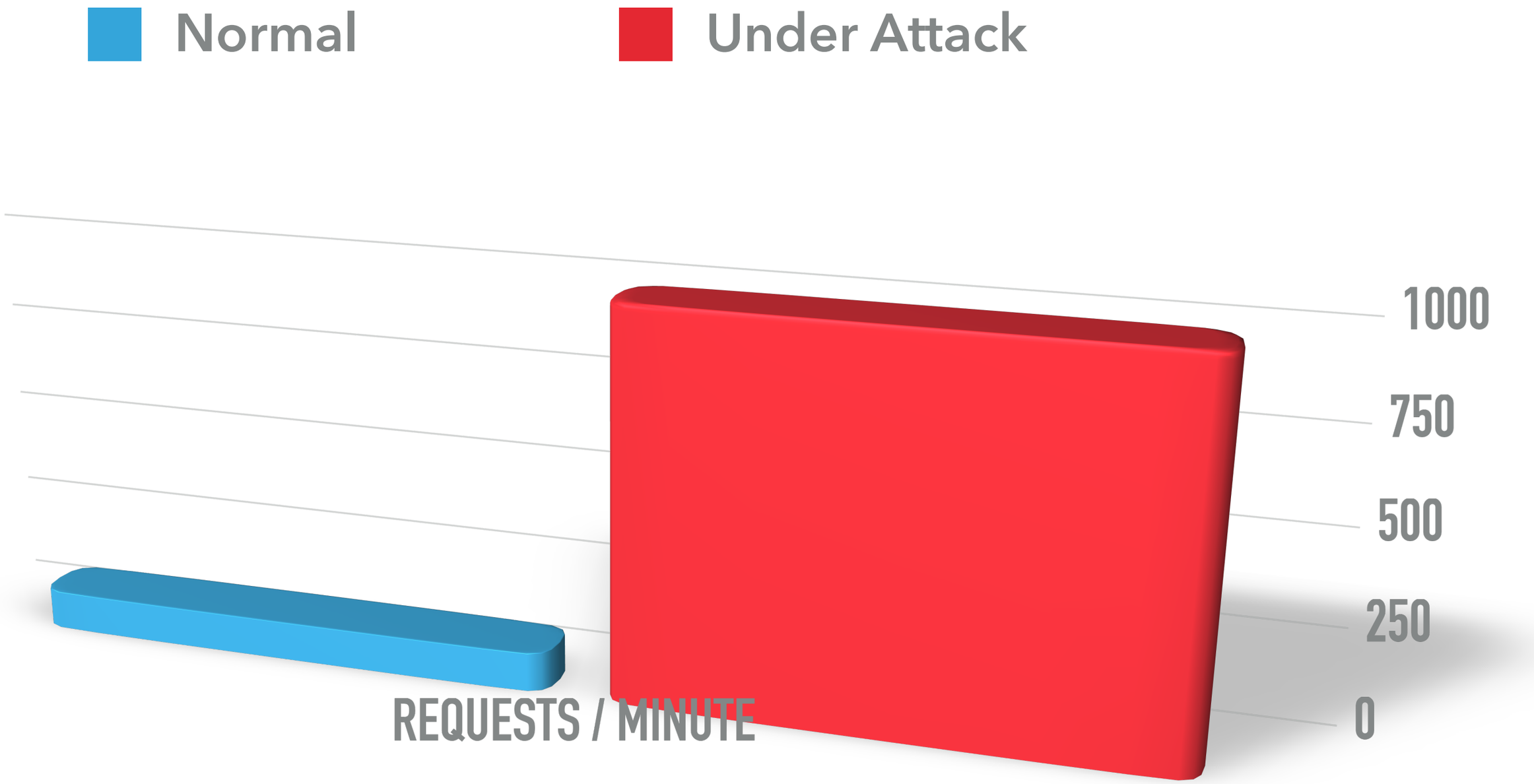
BUT WE ARE SAFE, RIGHT?

- We followed best practices
  - Strong Parameters
    - Bad
      - `User.update!(params[:user])`
    - Good
      - `User.update!(params.require(:user).permit(:name, :email)`

BUT WE ARE SAFE, RIGHT?

- We followed best practices
  - Parameterized queries
    - Bad
      - `User.where("email = '#{email}")`
      - `User.where("email = '%{email}'" % { email: email })`
    - Good
      - `User.where(email: email)`
      - `User.where("email = ?", email)`

REQUEST RATE



THE IMPROVEMENT

THROTTLE REQUESTS

- How often does a client need to make a request?
  - 10s of requests/minute
    - Reasonable
  - 1,000s of requests/minute
    - Unreasonable

THROTTLE REQUESTS

- We can limit the number of requests allowed by
  - A client
  - In a specific time window
- Different endpoints have different use profiles
  - Ideally, we can have different throttling parameters
- Our Barbican

WHAT DO WE NEED?

- Identify Clients
  - Rack::Request object provides the client IP address
- Remember clients requests (counts)
  - Store in memory?
    - For now...
- Define throttle parameters
  - Initializer

RACK::ATTACK

- Built by Kickstarter Engineering
- Ruby rack middleware for throttling abusive requests
- Keeps anomalistic request behavior in check
- Improves developer productivity and happiness!
  - Actual claim, check the blog post

APP SETUP

- Create a new rails application
  - `rails new sitting-duck`
- Initial Commit
  - `cd sitting-duck`
  - `git add .`
  - `git commit -m "Initial commit”`

APP SETUP

- Create a repository on GitHub
  - `sitting-duck`
- Push your app to GitHub
  - `git remote add origin https://github.com/bencarle/sitting-duck.git`
  - `git push -u origin master`

DEPLOYMENT SETUP

- Create a Heroku app
  - `sitting-duck`
- Deploy your app to Heroku
  - `git push heroku master`

CONTROLLER SETUP

- Create a Ducks controller
  - `rails generate controller Ducks index create`
- Setup routes
  - `resources :ducks, only: [:index, :create]`

RACK::ATTACK SETUP

- Add the rack-attack gem
  - `gem 'rack-attack'`
- Tell your app to use the Rack::Attack middleware
  - `config.middleware.use Rack::Attack`
- Add `rack-attack.rb` to `config/initializers`
  - `Rack::Attack.cache.store = ActiveSupport::Cache::MemoryStore.new`

THROTTLE

```
throttle('req/ip', :limit => 3, :period => 1.minutes) do |req|
  req.ip
end
```

THROTTLE

```
throttle('req/ip', :limit => 3, :period => 1.minutes) do |req|
  req.ip unless req.path.start_with?('/assets')
end
```

THROTTLE

```
throttle('req/ip', :limit => 3, :period => 1.minutes) do |req|
  req.ip unless req.get?
end
```

THE NEW REALITY

THROTTLED REQUESTS

- What have we accomplished?
  - Reduced the ability for an attacker to detect injection vulnerabilities
- We have NOT prevented an injection attack

THROTTLED REQUESTS

- What do we observe?
  - Bad API & script architecture will break
  - Script makes one request per object in series?
    - Not anymore
- Need better API design
  - Batch jobs

REFERENCES

- Rack::Attack
  - https://github.com/kickstarter/rack-attack
- Example Configuration
  - https://github.com/kickstarter/rack-attack/wiki/Example-Configuration
- Advanced Configuration
  - https://github.com/kickstarter/rack-attack/wiki/Advanced-Configuration
- Our sitting-duck app
  - https://github.com/bencarle/sitting-duck