## ActionController::Parameters .each returns an unsafe hash

Share: f in Y

TIMELINE

abuisman submitted a report to Ruby on Rails.

Nov 24th (5 years ago)

Rails 5.1.4

The goal of ActionController::Parameters 's permit method (strong parameters) is to prevent accidental trust in the parameters sent by the client. We can therefore not simply create a hash of all the parameters in the params without permitting them first. When we really want to do this there is the method to\_unsafe\_h , indicating the importance of controlling when an unsafe hash is returned. However, when we use \_\_each\_\_ on our parameters object, an unsafe hash is returned that includes all the keys and their values in a new hash:

```
Code 644 Bytes Wraplines Copy Download

1 params = ActionController::Parameters.new(city: 'Nijmegen', country: 'Netherlands', language: 'Dutch')

2
3 params.to_h

4
5 # ActionController::UnfilteredParameters: unable to convert unpermitted parameters to hash
6 # from ...lib/ruby/gems/2.4.0/gems/actionpack-5.1.4/lib/action_controller/metal/strong_parameters.rb:265:in 'to_h'

7
8 params.permit(:city)
9 => <ActionController::Parameters {"city"=>"Nijmegen"} permitted: true>

10
11 params.permit(:city).to_h
12 => {"city"=>"Nijmegen"}
13
14 params.to_unsafe_h
15 => {"city"=>"Nijmegen", "country"=>"Netherlands", "language"=>"Dutch"}
16
17 params.each {}
18 => {"city"=>"Nijmegen", "country"=>"Netherlands", "language"=>"Dutch"}
```

This behaviour is extra strange when contraste with how select works:

```
Code 141 Bytes

Wrap lines Copy Download

1 params.select { true }

2 => <ActionController::Parameters {"city"=>"Nijmegen", "country"=>"Netherlands", "language"=>"Dutch"} permitted: false>
```

Here you can see that select returns an instance of ActionController::Parameters that has permitted: false

## Impact

An attacker could find out about the accidental use of each in working with parameters in a controller and use this knowledge to send additional (more than provided in a form) parameters along and in this way circumvent authorisation checks.

```
Code 2:0 Bytes Wrap lines Copy Download

1  # controller:
2
3  def update
4  # Attacker has included the parameter: `{ is_admin: true }`
5  User.update(clean_up_params)
6  end
7
8  def clean_up_params
9
10  params.each { |k, v| SomeModel.check(v) if k == :name }
11  end
```

The example (admittedly simplified) above shows a possible scenario where a developer builds a method to do something with each param in a separate method after which he might expect his parameters to adhere to normal working [permitted: true/false]. Slightly unexpected behaviour that could cause security issues.

 $Biggest\ threat\ would\ seem\ to\ be\ to\ open source\ projects\ where\ attackers\ can\ survey\ the\ project's\ code.$ 

Ruby on Rails staff posted a comment.

Dec 5th (5 years ago)

Thank you for the report. Our team will investigate the issue and we will return back in a few days.

In mean time we ask you to not disclose this issue with anyone until we fully discard it as a security issue.

O-rafaelfranca (Ruby on Rails staff) added weakness "SQL Injection" and removed weakness "Cross-site Scripting (XSS) - Generic".

Updated Feb 28th (5 years ago)

O= rafaelfranca Ruby on Rails staff updated the severity from High (7.1) to Medium (6.5).

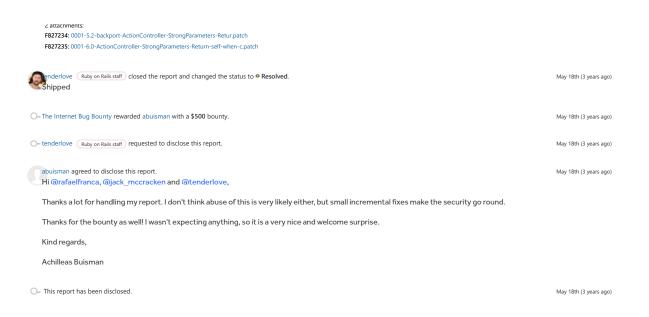
Dec 5th (5 years ago)

O= lasagna removed weakness "SQL Injection"

May 13th (3 years ago)

jack\_mccracken changed the status to • Triaged

May 13th (3 years ago)



 $\equiv$