Managing content security and samesite cookies

@oreoshake
@ndm

What is content security policy

Content-Security-Policy:

```
default-src 'none';
base-uri 'self';
block-all-mixed-content;
connect-src 'self' uploads.github.com...;
font-src github.githubassets.com;
form-action 'self' github.com gist.github.com;
frame-ancestors 'none';
frame-src render.githubusercontent.com;
img-src 'self' data: github.githubassets.com ...;
manifest-src 'self';
media-src 'none';
script-src github.githubassets.com;
style-src 'unsafe-inline' github.githubassets.com;
```

What is content security policy

Content security policy is hard

What is content security policy

CSP can be managed

Important features:

- Globally defined policy
- Dynamic policies
- Familiar, documented APIs

Model View Controller

Model
is the data

Controller
fetches and
manipulates
data

View
Displays
the data

application_controller.rb Raw

```ruby
class ApplicationController < ActionController::Base
  def add_csp_exceptions
    append_content_security_policy_directives(self.class::CSP_EXCEPTIONS)
  end
end
```

example_controller.rb Raw

```ruby
class ExampleController < ApplicationController
  CSP_EXCEPTIONS = {
    frame_src: ["foo.com"],
  }
  before_action :add_csp_exceptions, only: :show

  def index
    # doesn't get the additonal allowance
  end

  def show
    # gets foo.com appended to the current frame_src value
  end
end
```

other_controller.rb Raw

```ruby
class OtherController < ApplicationController
  def index
    if feature_enabled?
      append_content_security_policy_directives(connect_src: [THIRD_PARTY])
    end
  end
end
```

_reused_component.html.erb Raw

```erb
<%
  append_content_security_policy_directives(
    frame_src: [PAYMENT_PROCESSOR]
  )
%>

... code for payment UI
```

inline_script.html.erb Raw

```erb
<%= nonced_javascript_tag do %>
  // static JS goes here, please don't use dynamic javascript.
<% end %>
```

security_headers_dependency.rb Raw

```ruby
module SecurityHeadersDependency
  SecureHeaders::Configuration.override(:static_file_policy) do |config|
    config.csp = {
      default_src: [SecureHeaders::CSP::NONE],
      script_src: [SecureHeaders::CSP::SELF],
      style_src: [SecureHeaders::CSP::UNSAFE_INLINE],
      img_src: [SecureHeaders::CSP::SELF, SecureHeaders::CSP::DATA_PROTOCOL],
      connect_src: [SecureHeaders::CSP::SELF],
      form_action: [SecureHeaders::CSP::SELF],
      base_uri: [SecureHeaders::CSP::SELF],
    }
  end

  # Set Content-Security-Policy header for static files
  #
  # Returns nothing.
  def set_static_file_csp
    SecureHeaders.use_secure_headers_override(request, :static_file_policy)
  end
end
```

the_controller.rb Raw

```ruby
class TheController < ApplicationController
  def render_404(exception = nil)
    set_static_file_csp
    render "error/404"
  end
end
```

Samesite cookies

Prevent the browser from sending cookies across sites*,
making CSRF a lot harder

Samesite cookies

```
# Specify cookies SameSite protection level: either :none, :lax, or :strict.
#
# This change is not backwards compatible with earlier Rails versions.
# It's best enabled when your entire app is migrated and stable on 6.1.
# Rails.application.config.action_dispatch.cookies_same_site_protection = :lax
```

Samesite cookies

Are kinda broken in this weird transition

Samesite cookies

Fairly easy to configure

controller.rb Raw

```ruby
if current_user&.samesite_lax_enabled?
  SecureHeaders.use_secure_headers_override(request, :samesite_lax)
end
```

security_headers.rb Raw

```ruby
SecureHeaders::Configuration.default do |config|
  config.cookies = {
    samesite: {
      none: { only: ["saml_csrf_token", "saml_return_to"] },
    }
  }
end

SecureHeaders::Configuration.override(:samesite_lax) do |config|
  config.cookies = {
    samesite: {
      lax: { except: ["saml_csrf_token", "saml_return_to", "saml_csrf_token_legacy", "saml_return_to_legacy"] },
      none: { only: ["saml_csrf_token", "saml_return_to"] }
    }
  }
end
```

```
# persist the relay state digest for future validation
# Cookie attributes are additionally set by by app/controller/application_controller/security_headers_dependency.rb
# By default the :saml_csrf_token will have the SameSite=none attribute set from the secure_headers gem.
# The :saml_csrf_legacy cookie does not have the SameSite=none attribute set. This is to accomodate
# browsers that reject or mistreat cookies with the SameSite=none attribute. In particular, there is
# a bug that treats SameSite=None and invalid values as Strict in macOS before 10.15 Catalina and in iOS before 13.
cookies.encrypted[:saml_csrf_token] = saml_csrf_cookie(relay_state.digest)
cookies.encrypted[:saml_csrf_token_legacy] = saml_csrf_cookie(relay_state.digest)
```

May 23, 2019

Maintainer security advisories

We've released maintainer security advisories as a public beta. Maintainer security advisories allow open source maintainers to privately discuss, fix, and publish notices about security vulnerabilities in repositories.

GitHub may additionally create security alerts to all affected downstream repositories as appropriate.

Learn more about maintainer security advisories

Security Advisories

New draft security advisory

Privately discuss, fix, and publish information about security vulnerabilities in your repository's code.

0 Draft ✓ **2 Published** ✕ 0 Closed

Limited header injection when using dynamic overrides with user input moderate severity

GHSA-w978-rmpf-qmwg published on Jan 21 by oreoshake

Directive injection when using dynamic overrides with user input moderate severity

GHSA-xq52-rv6w-397c published on Jan 21 by oreoshake

Questions? Concerns? Comments?

Secure defaults, global configuration, and dynamic overrides are all you need to support rolling out these controls in a **development-friendly**, **reviewable**, and **secure** manner.