6 **The authenticity_token can be reversed and used to forge valid per_form_csrf_tokens for arbitrary routes**

Share:

TIMELINE

jregele submitted a report to **Ruby on Rails**. Nov 8th (3 ye

When `per_form_csrf_tokens` is set to `true`, each form should protected against CSRF with a unique token that is not predictable by an attacker.
The `per_form_csrf_token` is generated using a HMAC SHA-256 using a key that is exposed in a reversed `authenticity_token`. The `authenticity_token` is a Base6
encoding of

`one_time_pad | (one_time_pad XOR session[:csrf_token])`

Because the `one_time_pad` is the first half of the `authenticity_token`, it is not secret and an attacker can reverse the token to learn `session[:csrf_token]`.

From there the attacker can forge a hash for an arbitrary route (e.g. /articles/2):
`HMAC ( session[:csrf_token], "/articles/2#patch")`

To reproduce:

1. Have two Rails routes that accept only per form csrf tokens
2. Validate that the `authenticity_token` sent in the POST data returns an HTTP 422 when sent in the other form
3. Forge a per form token with the attached exploit script
   - the `authenticity_token` parameter is taken from the `<meta>` tag in the header of any page for the session
   - the `route` parameter is the action of the target HTML form (e.g /articles/2)
   - the `method` parameter is the value from `_method` parameter sent in the POST data (e.g. patch)
4. Take the forged token from the exploit script, URL-encode it, and send it as the `authenticity_token` in the POST data. For reliability, ensure that:
   - the route parameter matches the endpoint,
   - the `_method` parameter in the POST data matches what used for `method` in the exploit script
   - the forged `authenticity_token` in the POST data is properly URL-encoded

Impact

Exploitation allow an attacker to forge valid per-form CSRF tokens even in hardened situations where the global `authenticity_token` itself is not allowed.

For the attack to be successful, an attacker would need a valid global `authenticity_token`. This can be extracted out of a web page without any protections that cookies have (such as HTTP-Only). An attacker could leverage an XSS vulnerability to bypass per-form CSRF on unrelated pages. Because the token can be forged any form, code execution on a page without forms could still lead to attackers bypassing CSRF protections of forms related to password changes, deletion of data creation of new users, etc.

1 attachment:
F629421: perform_csrftoken_forgery.py

ktistai posted a comment. Nov 10th (3 ye
Hi @jregele

Thank you for your submission. Your report is currently being reviewed and the HackerOne triage team will get back to you once there is additional information to share.

Kind regards,
@ktistai

ktistai changed the status to **Needs more info**. Nov 11th (3 ye
Hi @jregele

Do you happen to have the code that you have used for testing? Also, any information regarding how to actually use it (installation and setup) would be really appreciated, as it would speed up the triage time.

Thanks,
@ktistai

jregele changed the status to **New**. Nov 12th (3 ye
Hi @ktistai,

Yes, I can give you detailed instructions. Please forgive me if they are overly detailed in case you are familiar with Burp Suite, but it may be the quickest way to actu demonstrate the vulnerability. I built a sample Rails app using the tutorial found here

https://guides.rubyonrails.org/getting_started.html

Unzip the attached archive which includes all the source code. From the root directory of the app, you will need to run
`rails db:migrate`

I also made a quick and dirty edit in request_forgery_protection.rb and set
`self.per_form_csrf_tokens = true`
This edit is in the Rails code itself. I'm sure there is a way to configure the application but this worked for the PoC. If this isn't set then the per-form CSRF tokens wo be checked at all.

From there, here are detailed instructions

1. Run `rails serve` and navigate to http://0.0.0.0:3000/articles/new

4. Create 2 articles. Title and text can be anything, but it is useful for them to be distinct (i.e. asdf/asdf and qwer/qwer)
5. Go to http://0.0.0.0:3000/articles/ and see the articles that have been created.
6. Use Burp Suite as an intercepting proxy (there is a free community edition). Intercepting traffic on localhost can be problematic, but loading the application at 0.0.0.0:3000 will work. (optional: Under Target -> Scope, under "Include in scope", click "Add" and enter http://0.0.0.0:3000/. Then Proxy -> Options -> Interce Client Requests, make sure "Intercept requests based on the following rules" is checked. Then make sure that the "is in target scope" rule is enabled. This will fi out all other requests from the proxy.)
7. Make sure proxy inteception is working based on your OS and Browser. Under Proxy -> Intercept, make sure the "Intercept is on" button is pressed. Load http://0.0.0.0:3000/articles/ in the browser. The request should be intercepted by Burp suite
8. Disable interception in Burp
9. click 'edit' for one of the articles.
10. Enable interception in Burp
11. On the web page, click 'Update Article'. The POST request will appear in the intercept window in Burp
12. In Burp proxy, right click the intercepted request and select "Send to Repeater"
13. Go the the repeater tab and see something like:

```
`POST /articles/1 HTTP/1.1
Host: 0.0.0.0:3000
Content-Length: 210
Cache-Control: max-age=0
Origin: http://0.0.0.0:3000
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.70 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,/;q=0.8,application/signed-exchange;v=b3
Referer: http://0.0.0.0:3000/articles/1/edit
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: csrftoken=9tlkrMsUDODKWNM9Qq3kbv6Pd84tHDu7mUIAWVwQExhwiD1guG55U38DfrkINYVg;
_blog_session=Qe%2FKflKKyab%2BJysm56C5YkRBtuwqRkdJNtwtxyY6FM4llpNxDru%2BijMX1hdSJwzRUMDRHvo2Wa5uWpWFHZpkChbgqB9zuiFoNegHSRrv
PW5p7fKWgUFBBfSfqrtcdRVSMrN48FBDRHa2TqQ6OYwBzuKbyWjsCbn%2BoG45hFdBfCwmzPbmg%2Bx5nqaWO6%2B7Gmv3002gueqhwxlPaB7xbCvpnPMlt
X%2BNygnxU2RXenVHDSSvAmxlAR8haUSQW0RWwV7%2F47ZVhARZylCbd8RfUNzuGW--hZ9kG6mEU8XQZElw--HmlncMiDDC7XsQQCyKHASw%3D%3D
Connection: close

_method=patch&authenticity_token=uNPIL0xLS%2BQuvXsY%2FoPYxujT%2FLZdWOvME%2FNJkYYFx3aLL1l6AJuIUMBFGlH7vio7x2u0W%2FbrrVcCbLsxV%
Sw%3D%3D&article%5Btitle%5D=zxcv&article%5Btext%5D=zxcvzxcv&commit=Update+Article`
```

14. Click Send.

15. The response pane should show an HTTP/1.1 302 Found in the top line. The bottom of the request will read:
    `<html><body>You are being <a href="http://0.0.0.0:3000/articles/1">redirected</a>.</body></html>`
    This is what a successful form post looks like.

16. Change the top line of the request to `POST /articles/2`, where the 2 here is a different number than the articles/1 in the previous request. If per-form CSRF t are enabled, the server will respond with a 422 message and an error page. If the server does not return a 422, per-form-csrf-tokens are not enabled. See my r about editing request_forgery_protection.rb. If that file has been successfully edited, restart the rails server.

17. Run the exploit to forge an `authenticity_token` for the request. Usage of the exploit script is as follows:
    `python perform_csrftoken_forgery.py <csrf-token from step 3> <route> <method>`
    This requires the csrf-token gathered in step 3. If the request that failed was `/articles/2`, use "articles/2" for the route. The method here is "patch" (no quot So if we are forging a token to edit /articles/2, we would run the exploit as follows
    `python perform_csrftoken_forgery.py <csrf-token from step 3> "articles/2" patch`

18. Copy the output token and in Burpsuite go to the Decoder tab. Paste the forged token into the pane, and then click "Encode as ..." and select URL.

19. Copy the URL encoded string and go back to the Repeater tab.

20. Replace the value of the authenticity_token with the URL encoded forged token. You can also change the title or article values, but make sure not to change an the %5D or %5B characters. Click send

21. The form will be accepted and the server will respond with a 302 response like it did in the first request. This means the forged token was accepted as valid.

1 attachment:
F632807: blog.zip

jregele posted a comment. Nov 12th (3 ye

Also, I wanted to add that the pentest where I discovered this, I found the client is passing authenticity_tokens and csrf-tokens around in URLs. This means that th token could be exposed by means other than XSS, such as in browser history, proxy and server logs, etc. If an attacker would to discover a token in any of these avenues, they could bypass all CSRF protections, even per-form csrf protections.

The actual vulnerable code is in `request_forgery_protection.rb` in the `masked_authenticity_token` method

`one_time_pad = SecureRandom.random_bytes(AUTHENTICITY_TOKEN_LENGTH) encrypted_csrf_token = xor_byte_strings(one_time_pad, raw_token) masked_token = one_time_pad + encrypted_csrf_token Base64.strict_encode64(masked_token)`

A more secure way to handle this would be to use an HMAC using the `raw_token` as the key. The one_time_pad could be replaced as random data that is then prepended to the hash. The random data could be exposed to the user, but the raw_token would remain secret to only the server.

ktistai posted a comment. Nov 16th (3 ye

@jregele,

ktistai posted a comment. Nov 16th (3 ye

LE: not quite finished, I need to enable editing on the articles.

ktistai changed the status to Needs more info. Nov 16th (3 ye

Hi @jregele

At `step 4, or 17 if the numbering would be right) Run the exploit to forge an authenticity_token for the request. Usage of the exploit script is as follo` am getting an error:

Code 580 Bytes Wrap lines Copy Dow

```
python perform_csrftoken_forgery.py "dqi3Es2zNAPek2IUZQQZS4FMWVwXT7Sjcu9mmcxMp%2FU6eUpv43dxUXxLXyQOOIESlSF78Iu9Vk%2BGLnSCONhyqg%3D%3D" articles/2 patc
Traceback (most recent call last):
  File "perform_csrftoken_forgery.py", line 49, in <module>
    session_csrf = reverse_authenticity_token(args.authenticity_token)
  File "perform_csrftoken_forgery.py", line 23, in reverse_authenticity_token
    real_csrf_token = xor_bytes(list(one_time_pad), list(masked_csrf_token))
  File "perform_csrftoken_forgery.py", line 11, in xor_bytes
    assert len(key) == len(buf)
AssertionError
```

What did I do wrong?

Thanks,
@ktistai

jregele changed the status to New. Nov 17th (3 ye

The authenticity token is URL-encoded, so there are additional bytes. You could use the Decoder tab in Burp, and run Decode as -> URL. That will give you the bas encoded version.
I got it to work with this:
`perform_csrftoken_forgery.py dqi3Es2zNAPek2IUZQQZS4FMWVwXT7Sjcu9mmcxMp/U6eUpv43dxUXxLXyQOOIESlSF78Iu9Vk+GLnSCONhyqg== articles/2 patch`

Let me know if you have any more questions. I will have limited connectivity the next 2 days, but after wednesday should be back to normal.

Best,
Justin

ktistai changed the status to Needs more info. Updated Nov 18th (3 ye `self.per_form_csrf_tokens = false` and modified it to be `true`. Is there anything else?

Code 2.35 KiB Wrap lines Copy Dow

```
Started GET "/articles" for 192.168.178.10 at 2019-11-18 13:53:33 +0100
Cannot render console from 192.168.178.10! Allowed networks: 127.0.0.0/127.255.255.255, ::1
Processing by ArticlesController#index as HTML
  Rendering articles/index.html.erb within layouts/application
  Article Load (0.2ms)  SELECT "articles".* FROM "articles"
  ↳ app/views/articles/index.html.erb:10
  Rendered articles/index.html.erb within layouts/application (Duration: 17.9ms | Allocations: 1683)
[Webpacker] Everything's up-to-date. Nothing to do
Completed 200 OK in 83ms (Views: 82.0ms | ActiveRecord: 0.2ms | Allocations: 5101)


Started GET "/articles/2/edit" for 192.168.178.10 at 2019-11-18 13:53:34 +0100
Cannot render console from 192.168.178.10! Allowed networks: 127.0.0.0/127.255.255.255, ::1
Processing by ArticlesController#edit as HTML
  Parameters: {"id"=>"2"}
  Article Load (0.1ms)  SELECT "articles".* FROM "articles" WHERE "articles"."id" = ? LIMIT ?  [["id", 2], ["LIMIT", 1]]
  ↳ app/controllers/articles_controller.rb:15:in `edit'
  Rendering articles/edit.html.erb within layouts/application
  Rendered articles/_form.html.erb (Duration: 21.4ms | Allocations: 724)
  Rendered articles/edit.html.erb within layouts/application (Duration: 24.5ms | Allocations: 1570)
[Webpacker] Everything's up-to-date. Nothing to do
Completed 200 OK in 47ms (Views: 44.2ms | ActiveRecord: 0.1ms | Allocations: 5501)


Started PATCH "/articles/2" for 192.168.178.10 at 2019-11-18 13:54:06 +0100
Cannot render console from 192.168.178.10! Allowed networks: 127.0.0.0/127.255.255.255, ::1
Processing by ArticlesController#update as HTML
  Parameters: {"authenticity_token"=>"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABtuqgNMsUiVNbYzoh/C9ZCHULmakV28BVI7rna5Pyapg==", "article"=>{"title"=
Can't verify CSRF token authenticity.
Completed 422 Unprocessable Entity in 1ms (ActiveRecord: 0.0ms | Allocations: 676)



ActionController::InvalidAuthenticityToken (ActionController::InvalidAuthenticityToken):

actionpack (6.0.1) lib/action_controller/metal/request_forgery_protection.rb:217:in `handle_unverified_request'
actionpack (6.0.1) lib/action_controller/metal/request_forgery_protection.rb:249:in `handle_unverified_request'
actionpack (6.0.1) lib/action_controller/metal/request_forgery_protection.rb:244:in `verify_authenticity_token'

```

These are my actual logs at the latest try. I am always getting that 422 Error.

Thanks,
@ktistai

jregele changed the status to New. Nov 20th (3 ye

Yes changing that line in `request_forgery_protection.rb` looks like it worked successfully. Odd it wasn't working for you though. Were you using the same token fr the previous session in the new session? I've been traveling the past few days but should be to my final destination by later today. It might be useful to do a call so can screen share and go through the steps. I got it to work relatively quickly, but it does involve some crypto and hashing, so if there is 1 bit off, it won't work. Let m know if and when you're available to do that.

Best,
Justin

ktistai changed the status to Needs more info. Nov 21st (3 ye

@jregele

I do agree, because of the complexities involved any small mistake may make it fail. We can use keybase, where my id is `ktistai`.

Thanks
@ktistai

jregele changed the status to New. Nov 21st (3 ye

I figured out what is most likely going wrong. There are 2 things.

1. the route needs to have a slash in the beginning, so `/articles/2` instead of `articles/2`. I stepped through the code in a debugger to make sure.
2. The token that needs to be fed to the python script needs to come from the `<meta>` tag with the name `csrf-token` at the top of the page source. It isn't URL encoded, which is why I never had to URL decode before using the script. I think you were pulling the token from the form submission request. Once per-form tokens are enabled, the `authenticity_token` in the form will already be hashed. So what the python script does it reverse the `csrf-token` in the `<meta>` tag t the internal token, and then hashes it with the new route.

I'm going to make a video of it tomorrow to show the steps because jet lag is killing me at the moment.

jregele changed the status to New. Nov 23rd (3 ye

Attached is the demo video

1 attachment:
F642052: demo.zip

ktistai changed the status to Triaged. Nov 25th (3 ye

Hello @jregele

Thank you for your submission! We were able to validate your report, and have submitted it to the appropriate remediation team for review. They will let us know t
final ruling on this report, and when/if a fix will be implemented. Please note that the status and severity are subject to change.

Regards,
@ktistai

ack_mccracken posted a comment. May 13th (3 ye

Hi @jregele,

I've prepared the following patches and advisory to address this issue. Would you mind taking a look?

3 attachments:
F827086: 732415_advisory.txt
F827087: 0001-5.2-backport-HMAC-raw-CSRF-token-before-masking-it-s.patch
F827088: 0001-6.0.x-HMAC-raw-CSRF-token-before-masking-it-so-it-cannot-b.patch

enderlove Ruby on Rails staff closed the report and changed the status to Resolved. May 18th (3 ye

Shipped

jregele posted a comment. May 18th (3 ye

Hi @jack_mccracken, of course would be happy to take a look. Let me rebuild my environment and patch rails and I will get back to you.