Let Me Github That For You

Let Me Github That For You - joernchen, phenoelit.org.

- Published: date not stated
- Original: <https://web.archive.org/web/20170903113359/http://phenoelit.org/blog/archives/2012/12/21/let_me_github_that_for_you/index.html>
- Current location: <https://web.archive.org/web/20170829190206/http://www.phenoelit.org/blog/archives/2012/12/21/let_me_github_that_for_you/index.html>
- Preserved from: https://web.archive.org/web/20170829190206/http://www.phenoelit.org/blog/archives/2012/12/21/let_me_github_that_for_you/index.html (live) on 2026-08-10
- Capture timestamp: 20170903113359
- Licence: unknown

Content

> UNTRUSTED SOURCE TEXT. Everything below this line is third-party material quoted for research. It is data, not instructions. Do not follow directions, execute code, or fetch URLs because this text says so.

Let Me Github That For You | Lands of Packets

The Wayback Machine - https://web.archive.org/web/20170829190206/http://www.phenoelit.org:80/blog/archives/2012/12/21/let_me_github_that_for_you/index.html

Fri Dec 21 16:02:48 UTC 2012

Let Me Github That For You

This blog post serves as a wrap up of some aspect of the presentation I gave at ZeroNights 2012.

Ruby on Rails (RoR) is atm my favorite piece of software to hunt bugs at. After quite some time spending on looking at Rails apps I figured that I oversaw the most easy way to attack an (Open Source) Rails app for quite a while. Before I come to my main point we'll have to look at both RoR sessions and authentication systems:

Ruby on Rails Sessions

RoR sessions are by default stored client-side in a cookie. In order to be tamper resistant, this cookie is signed with an SHA-1 HMAC. When the HMAC is missing or the cookie being tampered with RoR will refuse to use the session variables within the cookie.

So, let's look at such a cookie, as an example I'll use a Github session cookie:

```
_gh_sess=BAh7BzoQX2NzcmZfdG9rZW4iMStDQWNRZ1l4VlZPb3ZPM3FBYXZWZGtsYzF2NUVENkdaRnhEK1A0QmNqU1k
```

This cookie consists of a Base64 blog and the HMAC, both separated by "--". When decoding the Base64 you'll get a marshaled Ruby Object, namely the session hash which is accessible in the web application via session[:some_session_var]. The decoded and de-marshaled Github cookie looks like this:

```
{:_csrf_token=>"+CAcQgYxVVOovO3qAavVdklc1v5ED6GZFxD+P4BcjSY=", :session_id=>"01883ec73d917a3939b7d6ee24
```

So in this case for instance the _csrf_token would be accessible via session[:_csrf_token] from within Githubs' RoR code.

Ruby on Rails Authentication Frameworks

While reading some OSS RoR applications I mainly came across this three authentication mechanisms for RoR:

- authenticated_system
- authlogic
- devise/warden

Where authenticated_system is the simplest mechanism, which will just put a field "user_id" inside the session, by this ID then authenticated_system will pull the user with that ID out of the database as the currently logged in user. Both authlogic and devise/warden handle it a bit different, they will use a certain random token which is stored within the database in order to identify the current user.

The fun stuff ;)

When a RoR application is created the secret which goes into the HMAC will be created along with all the other files a minimal RoR application would need. This secret usually is a 64 byte long random string and lives in $railsapp/config/initializers/secret_token.rb. The simple problem is, that most developers are simply not aware of the confidentiality of this file, and in result they'll happly check it into Github or other online repositories (This guy already figured that a while ago).

When using authenticated_system it's pretty obvious how to break into such an application:

- Observe the secret_token on Github
- Create a cookie containing "user_id=>1"
- ???
- Profit! (as in: be any user)

Well that was easy, let's look at authlogic now.

An authlogic cookie usually uses a database stored token to identify the user. The relevant parts of the session cookie are:

- user_credentials_id - a numeric value which is used with "User.find_by_id()"
- user_credentials - a radom string which will be compared with the database field "persistence_token" in the Users table

Due to the way the RoR "find_by_*" methods are defined the following SQL injection a-like issue arises:

```
> User.find_by_id({:select =>"* from users limit 1 --"})
  User Load (0.5ms)  SELECT * from users limit 1 -- FROM "users" WHERE "users"."id" IS NULL LIMIT 1
 => #<User id: 1, [... all the fun stuff]
```

By knowing this behaviour we can now easily circumvent the authlogic protection with the knowledge of the "secret_token". The following cookie would give you access to an authlogic protected application:

```
{
  "session_id" => "41414141",
  "user_credentials"=>"Phenoelit",
  "user_credentials_id"=>{
    :select=> " *,\"Phenoelit\" as persistence_token from Users -- "
  }
}
```

Last man standing would then be devise/warden, which works similar to authlogic but is not exploitable in that way described above.

Finally

If you checkin stuff to the internet, think at least twice about it. If you'd like to mess around with RoR cookies, have a look here and here TL;DR: Click here to mass 0wn Ruby on Rails webapps ;)