# Tequila

*Writing Clients*

Claude Lecommandeur

EPFL - KIS

claude.lecommandeur@epfl.ch

## Table of Contents

Overview....3
General principles....4
Steps....5
Step 2 : Client tries to find a key value either in the URL or in a cookie....5
Step 3 : Create the request on the server....5
Step 4 : Get the unique request key....5
1.Redirect the user to Tequila....6
2.Just wait for User to come back....6
3.Still wait....6
4.Welcome User and validate the key....6
5.Create the session....7
Information to give to the server....9
service....9
request....9
require....9
wish....9
urlacces....9
allows....9
belongs....9
language....9
Obtaining server configuration....10
Signed URL mode....12

# Overview

Services providers that want to use the Tequila server for authenticating their users have to be be able to dialog with this server. There is already high level interfaces for several languages that are bundled in the standard Tequila distribution : Perl, PHP, Java, and an Apache module that encapsulate the dialog with the server.

Writing such interfaces is not difficult. The purpose of this document is to help you in this task. All examples are in Perl for two reasons. Perl is very expressive, well written programs are easy to read and understand. And this the language I belove.

The Tequila server is a CGI server, so speaking with it is done exclusively via URL calls and redirections. Applications can either call a Tequila URL and read the answer, or redirect the user's browser to the server.

In all this document the string *tequila-url (in italic)* will design the URL of the Tequila server. Generally it has the form : https://tequila.*your_domain*/cgi-bin/tequila, but it is not necessary. If you are using Tequila inside Apache mod_Perl, it will probably be different. One of the job of the Tequila client is to securely devise the name of its Tequila server.

# General principles

There are 3 actors in the Tequila comedy. A people with a browser (**User**), tries to access a protected Web application (**Application**). The Client uses the service of a Tequila server (**Tequila**) to authenticate the User and get some information about him. The scenario is the following :

1. **User** access an URL on **Application**.
2. **Application** tries to find a key value either in the URL or in a cookie.
3. If there is no key, **Application** calls an URL on **Tequila** with the 'createrequest' command. It sends along the description of the request. If **Application** is a resource (see below), it just sends its resource name. In the other case, it sends all the characteristics of the request. There is many of them, the most important are the URL where to redirect the user after authentication (urlaccess), the attributes values the client wants to know (request), the service name (service), etc...
4. **Tequila** stores the attributes of the request and responds with a random hexadecimal request key.
5. **Application** reads this key and redirects **User's** browser to **Tequila** URL plus the request key.
6. **Tequila** recognize the key, fetch the corresponding authentication request and does what it is meant to do, authenticate **User**, fetch attributes values about him, check if constraints are met, and if everything is OK, redirects **User** to 'urlaccess' plus the request key as a parameter. Optionally Server deposit a specially crafted cookie it can recognize later in **User**'s browser.
7. **Application** will see **User** comes back, but this time, it finds the request key it was waiting for.
8. It first tries to see it there is a session already opened for this key.
9. If there is, **User** is given access to the services of **Application**.
10. But just after **User** returns from **Tequila**, there is no such session. So, **Application** calls another URL on **Tequila** with the command 'fetchattributes', along with the request key.
11. **Tequila** sends back the values of the attributes relative to **User** (username, name, etc...).
12. **Application** creates a Session for **User** with all the attributes values stored in it for future use.
13. **Application** also set a cookie in **User's** browser with a session key.

We are trying to describe here the Client's job. So we have to fill steps 2, 3, 5, 7, 8, 9, 10, 12 and 13. Managing sessions is not mandatory but is is much more efficient. If you don't, you will go to **Tequila** for every page **User** asks for.

# Steps in detail

## Step 2 : Client tries to find a key value either in the URL or in a cookie.

Generally these kind of information is stored in the environment (QUERY_STRING, HTTP_COOKIE), or in the standard input of the script. Each and every language has it’s own way to access these data, but it is not very hard to do. First tine **User** comes in, there is no key at all. We will first go to this branch.

## Step 3 : Create the request on the server.

**User** shows up requesting some protected service. **Application** should call the URL : t*equila-url*/createrequest with the *POST* method. The body of the HTTP request will be filled with all the request attributes, one per line. Example :

```
POST /cgi-bin/tequila/createrequest HTTP/1.0
Host: your_tequila_server
Content-length: the_actual_request_length

urlaccess=http://myhost.mydomain/myapp
service=My service
request=name,firstname
require=group=somegroup
allows=category=guest
```

Use either HTTP or HTTPS for this request since there is not much secret about it. To build '*urlaccess*', we generally use simply the caller's URL :

```
  my $qs = $ENV {QUERY_STRING};
  my $pi = $ENV {PATH_INFO};
  my $me = $ENV {SCRIPT_NAME};
  my $us = $ENV {SERVER_NAME};
  my $urlaccess = "h ttp://$us$me/$pi
```

## Step 5 : Get the unique request key.

Now, read the response from **Tequila**

```
my $line = <SOCK>;
my ($status) = ($line =~ / (\d*) /); # should be HTTP/1.x 200 OK
if ($status != 200) {
  error ("Bad connection to local Tequila server ($server), server response is $line");
}
while (<SOCK>) { # skip headers
  last if /^[\r\n]*$/;
}
my $requestkey;
while (<SOCK>) { # body
  chomp;
  next if /^$/;
  $requestkey = $1 if /^key=(.*)$/;
}
close (SOCK);
```

Now you should have the request key. If not, there was an error some where you must dealt with.

You have just obtained a beautiful brand new request key, redirect **User** to **Tequila** with it. For this just write the Location command to **User'**s browser.

```
Location: tequila-url/requestauth?requestkey=$requestkey
```

## Step 10: User comes back with a brand new key (no session associated).

**User** has finally come back to *urlaccess*. But don't hold your breath, Sometimes **User** never comes back, he can abandon authentication, have lost his password, or refuse to disclose the value of a requested sensitive attribute.

At this stage we have no proof that the key handed by **User** has been authenticated by **Tequila**. We have to ask. So we open a socket to **Tequila** host (like in Step 3), ans POSTs this :

```
POST /cgi-bin/tequila/fetchattributes HTTP/1.0
Host: your_tequila_server
Content-length: the_actual_request_length

key=the_unique_key
```

Read the result status :

```
  my $line = <SOCK>;
  my ($status) = ($line =~ / (\d*) /); # HTTP/1.1 200 OK
  if ($status == 451) { # Invalid key.
    return;
  }
  elsif ($status != 200) {
    error ("Tequila:validatekey error status from server : $line");
  }
```

Only a status of 200 is OK. 451 means invalid key, everything else is mysterious.

Skip headers :

```
while (<SOCK>) { # skip headers
  last if /^[\r\n]*$/;
}
```

Get the user's attributes filled by **Tequila** :

```
my ($org, $user, $host, $key, %attrs);
while (<SOCK>) { # body
  chomp;
  next if /^$/;
  if (/^([^=]*)=(.*)$/) {
    my  $name = $1;
    my $value = $2;
    if    ($name eq  'org') { $org  = $value; }
    elsif ($name eq 'user') { $user = $value; }
    elsif ($name eq 'host') { $host = $value; }
    elsif ($name eq  'key') { $key  = $value; }
    else { $attrs {$name} = $value; }
  }
}
close (SOCK);
```

'*org*' is the organization of **User**, It is really important only when your Tequila server is part of set of mutually trusting servers.

'*user*' is the user name of **User.**

'*host*' is the host running the browser **User** authenticated from. It is up to you to check if it is identical to the host from where the request comes from.

'*key*' is the request key, it should be identical to the request key you handed to **Tequila**.

## Step 12: Create the session.

Finally, create a new session if necessary. There is many ways to create sessions, the simplest being to use files with the session key as name :

```
unless (-d $sessiondir && -w $sessiondir) {
  $self->error ("Tequila:createsession: Session directory $sessiondir doesn't ".
                          "exist or not writable.");
}
unless (open (SESSION, ">$sessiondir/$key")) {
  $self->error ("Tequila:createsession: Unable to open session file ".
                          "($sessiondir/$key) : $!");
}
print SESSION "org=$org\n",
              "user=$user\n",
              "host=$host\n";
foreach my $attr (keys %attrs) {
  my $value = $attrs {$attr};
  $value = "\\\n" . $value . "\n" if ($value =~ /[\n\r]/);
  print SESSION "$attr=$value\n";
}
close (SESSION);
```

If your application is running on a servers farm, you will face the problem of sharing sessions between several hosts. You can use NFS files or a database (Mysql) but you loose redundancy. If you want do retain redundancy, you can use the Distributed Session Manager (DSM) in the Tequila distribution. It uses UDP to multicast sessions in a set of servers.

## Step 13: Setting a cookie.

```
sub depositcookie {
  my ($self, $cook, $value) = @_;
  return unless $cook;
  my $date = gmtime (time + $self->{sessionmax});
  my ($day, $month, $daynum, $hms, $year) = split (/\s+/, $date);
  my $expires = sprintf ("%s %02d-%s-%s %s GMT", $day, $daynum, $month, $year, $hms);
  if ($self->{cookiepolicy} eq 'session') {
    print qq{Set-Cookie: $cook=$value; path=/;\r\n};
  } else {
    print qq{Set-Cookie: $cook=$value; path=/; expires=$expires;\r\n};
  }
}
```

You can choose your cookie policy. I think session cookies are better, but I have no proof.

## Step 13: Is there a session already opened for this key

Just go to your sessions directory (or wherever you store your sessions) and see if there is a session with the key you got. If yes, read it, it is OK.

## Requests parameters.

You have to the to the Tequila server what kind of request you want. The available parameters are :

- **resource**

  A string identifying the resource name. It can be used only if your application is known to the server as a resource. In this case, no other parameters can be used, all are already known to the server.

- **service**

  A string identifying the service. It will be displayed at the top of the login window.

- **request**

  The list of attributes you want to obtain about the user.

- **require**

  The filter you want to impose on the user's attributes. It is a parenthesized boolean expression with atomic members of the form : *attr1=value1*, or attr1. In the former case, attribute *attr1* of the user must have the value *value1* among its set of values (remember that attributes can be multi-valued). In the latter form, attribute *attr1* must be present and not null.

- **wish**

  Almost the same as request, except that is not an absolute requirement. If one the attributes in the wish list is sensitive, the user will be asked if he want to give out the value, he can refuse, in which case the value will not be set by the server.

- **urlacces**

  The URL where to redirect the user after successful authentication and constraints checking. It will appended with the unique server generated key.

- **allows**

  In a certain sense, this the contrary of '*require*'. By default, the Tequila server impose default restrictions on certain attributes values. Using allows can lift some (or all) of these restrictions.

- **language**

  Language to use in the interaction with the user (login window, errors, ...). The default server's language is set in the server's configuration files. The user will still be able to change the language.

## Obtaining server configuration

The application that wants to obtain details on the server has to call the URL : t*equila-url*/getconfig. The result is something like this :

```
Organization: EPFL
Server: tequila.epfl.ch
Domain: epfl.ch
Manager: claude.lecommandeur@epfl.ch
Cookies: on
Support certificates: Yes
Default language: fr
Supported user attributes : name firstname statut classe email title unit where office
phone username uniqueid unixid groupid group org categorie allunits unitid cf
Server certificate:
-----BEGIN CERTIFICATE-----
MIICmTCCAgKgAwIBAgICBCwwDQYJKoZIhvcNAQEEBQAwgYcxCzAJBgNVBAYTAkNI
MQOwCwYDVQQIEwRWYXVkMREwDwYDVQQHEwhMYXVzYW5uZTENMAsGA1UEChMERVBG
TDElMCMGA1UEAxMcRVBGTCBDZXJ0aWZpY2F0aW9uIEF1dGhvcml0eTEgMB4GCSqG
SIb3DQEJARYRY2VydC1hdXRoQGVwZmwuY2gwHhcNMDQwNzE5MDgwOTIOWhcNMTIw
MTEOMDgwOTIOWjCBhDELMAkGA1UEBhMCQ0gxDTALBgNVBAgTBFZhdWQxETAPBgNV
BAcTCExhdXNhbm5lMQOwCwYDVQQKEwRFUEZMMRgwFgYDVQQDEw9OZXF1aWxhLmVw
ZmwuY2gxKjAoBgkqhkiG9w0BCQEWG2NsYXVkZS5sZWNvbW1hbmRldXJAZXBmbC5j
aDCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAzQKCOI6nhOChTgdasIdtXSyE
jAyp1UXKTnnTcOiuRD7Cb9UOdmHVIq8K9wu48J/3Tshqwml7InGJflw4eOWH+FVZ
/mlbX4saRoJBy1V8xsYVoDKC/7IrSy+kDAw9XyFrNSCOrbwv3Cpk9pv5+kg/NveC
x6zBgyfLk9aRmElJI98CAwEAAaMVMBMwEQYJYIZIAYb4QgEBBAQDAgZAMA0GCSqG
SIb3DQEBBAUAA4GBAG6U/MSUGWJO7wOIRT6hpOUDrhOKYqxenDzcGkmx+2Q7xlVS
bHT/UXXJJeQJ2zSnAb/DRniXsqmXD1GM/oTvmUEacq6ufwa7XIpd7+Hg+62E28wn
p1KC8t3qR8W1oH3swxgyEMJW6KBHSZ+SI439pwdlcc6kdj4tlSiLmEemtnFH
-----END CERTIFICATE-----
```

Most attributes are self-explaining :

| | |
|---|---|
| **Organization** | The name of the local organization. This is the name that will be returned to the client in the 'org' attribute. |
| **Server** | The name of the server. |
| **Domain** | The Internet domain name. |
| **Manager** | Email of the server manager. |
| **Cookies** | Does this server use cookies to memorize users. |
| **Support certificates** | Does this server understands SSL certificates to authenticate users. |
| **Default language** | Default server language. |
| **Supported user attributes** | The most important attribute : the list of attributes supported by the server. Clients can request values and impose filters about these attributes. |
| **Server certificate** | The SSL certificate of the server. |