# WebServices API - Authentication

## Cookies

Explanation for how to manage/handle Cookie elements to prevent from making new WSSESSIONID with each request. (can be cited in each section after creating a session)

You need to capture the value supplied from the server of your initial GET /login.xml request and use that same value for 'WSSESSIONID' for all subsequent requests of that same session. This is what our servers use to identify a specific session.
Whether you store it locally in your running application or in a cache store elsewhere, the crucial point is that value provided is used for all subsequent requests. If the 'WSESSIONID' cookie is not present or does not match the first request you make against the API, the request will be treated as part of a new session and you will receive a "401 Unauthorized" response when trying to send your response body with POST /login.xml.

You might notice two other cookies: "BIGipServerp" and "Blaze". These are not added or used for application functionality and you can safely ignore them.

In all of these examples, we will be using the following values for username and password.

- username = **25livedemo**
- password = **CollegeNETTEST1**

## Basic Authentication

Basic authentication is a simple authentication scheme built into the HTTP protocol that can be handled by the browser. The client sends HTTP requests with the `Authorization` header that contains the word `Basic` word followed by a space and a base64-encoded string `username:password`. Because base64 is easily decoded, Basic authentication should only be used together with other security mechanisms such as HTTPS/SSL. This type of auth can also be used by scripts/code to negotiate authentication.

1. A client requests access to a protected resource.
2. The web server returns a dialog box that requests the user name and password.
3. The client submits the user name and password to the server.
4. The server authenticates the user in the specified realm and if successful, returns the requested resource.
   a. The **authentication** parameter '**realm**' is **REQUIRED and must be supplied in the response back TO the server.**
   b. The HTTP `WWW-Authenticate` response header defines the authentication method that should be used to gain access to a resource.

**GET to /run/login.xml to create WSESSIONID**

```
HEADER
WWW-Authenticate: Basic realm="R25 WebServices", charset="UTF-8"

HTTP
GET /r25ws/wrd/<instance>/run/login.xml HTTP/1.1
Host: webservices.collegenet.com
Authorization: Basic MjVsaXZlZGVtbzpDb2xsZWdlTkVUVEVTVDE=   <-- this is the hashed value for the username and password.
Cookie: Blaze=YNJkHw4b@s27bUhDOB5iTQAAFWI; WSSESSIONID=C02F3F9276F394FEF06EC8E1079748C6; BIGipServerp-java.webservices-web.collegenet.com=2250770186.36895.0000
```

## Challenge Response Basic Authentication

For an LDAP server to properly check a supplied password, it must be received as clear-text. There is not a mechanism setup to un-digest a password supplied from using the HTTP Digest process, therefore, the HTTP Basic process is used when LDAP is enabled. If security concerns exist, make sure to use an SSL enabled port for the connection between the WebServices/25Live application server and your contact directory. This is typically port 636.

When using login.xml with LDAP authentication, a hybrid challenge/response-basic method is used. Instead of a challenge string, Web Services responds with a template for Basic authentication:

1. Make a **GET** request to **login.xml**

**login.xml GET response**

```
<?xml version="1.0"?>
<r25:login_challenge pubdate="2006-03-10T11:02:00" xmlns:r25="http://www.collegenet.com/r25">
  <r25:login>
    <r25:challenge>Basic realm="R25 WebServices"</r25:challenge>
    <r25:username/>
    <r25:response/>
  </r25:login>
</r25:login_challenge>
```

2. Construct the response, clear out the challenge, and post your response. Formulate the response as you would with HTTP Basic authentication. In the response tag, put Basic in front of the response as below.

3.

**login.xml POST request payload**

```
<?xml version="1.0"?>
<r25:login_challenge pubdate="2006-03-10T14:10:24" xmlns:r25="http://www.collegenet.com/r25">
  <r25:login>
    <r25:challenge />
    <r25:username>25livedemo</r25:username>
    <r25:response>Basic bmF0aGFuOm5hdGhhblBVRkYAAAA</r25:response>
  </r25:login>
</r25:login_challenge>
```

4.

**login.xml POST response payload**

```
<?xml version="1.0"?>
<r25:login_response xmlns:r25="http://www.collegenet.com/r25" xmlns:xl="http://www.w3.org/1999/xlink" pubdate
="2011-12-01T13:58:26-08:00">
  <r25:login>
    <r25:message>Login successful</r25:message>
    <r25:success>T</r25:success>
    <r25:user_type>r25</r25:user_type>
    <r25:user_id>5486</r25:user_id>
    <r25:username>r25demo</r25:username>
    <r25:contact_name>Demo, R25</r25:contact_name>
    <r25:security_group_id>31</r25:security_group_id>
    <r25:security_group_name>Users</r25:security_group_name>
    <r25:login_url></r25:login_url>
    <r25:logout_url>https://r25.school.edu/25live/data/run/logout.xml</r25:logout_url>
  </r25:login>
</r25:login_response>
```

Challenge Response Authentication

Make a **GET** request to **login.xml.** You can choose to avoid HTTP authentication all together, if you use the challenge response system presented by the login.xml service. The process to use this is outlined below:

1. **GET** *https://webservices.collegenet.com/r25ws/wrd/<instance>/run/login.xml*

**login.xml GET request payload**

```
<?xml version="1.0" encoding="utf-8"?>
<r25:login_challenge xmlns:r25="http://www.collegenet.com/r25" xmlns:xl="http://www.w3.org/1999/xlink" pubdate
="2021-06-15T14:24:15-07:00" engine="sws">
   <r25:login>
      <r25:challenge>f5eea272958b21d26a3bf3a649bd31b1</r25:challenge>
      <r25:username/>
      <r25:response/>
   </r25:login>
</r25:login_challenge>
```

2. Calculate the "response string" following this formula: **MD5(MD5(password):<r25:challenge value>)**

- Create an MD5 hash of the user's password.
  - Ex: CollegeNETTEST1=8605492c6d7818728ad730c3834ba04d
- Append a colon and the challenge string from the initial GET of login.xml to this password hash.
  - Ex: 8605492c6d7818728ad730c3834ba04d:ecb4a7f2a7c10ac2411c7db4d557ecc6
- Generate **another** MD5 hash of the string created in in part b.
  - Ex: 1fb6b3c34f9a590f9555a51f0ed9e3ab

- Use value from part c in the <r25:response> line of the XML body.

3. **POST** *https://webservices.collegenet.com/r25ws/wrd/<instance>/run/login.xml* with the payload containing the <r25:response> to login.xml

**login.xml POST request**

```
<?xml version="1.0" encoding="utf-8"?>
<r25:login_challenge xmlns:r25="http://www.collegenet.com/r25" xmlns:xl="http://www.w3.org/1999/xlink" pubdate
="2021-06-15T14:24:15-07:00" engine="sws">
  <r25:login>
    <r25:challenge />
    <r25:username>25livedemo</r25:username>
    <r25:response>b4fe7f5591a4cd287b4500eae887ebf1</r25:response>
  </r25:login>
</r25:login_challenge>
```

4. Successful response from login.xml

**login.xml POST response**

```
<?xml version="1.0" encoding="utf-8"?>
<r25:login_response xmlns:r25="http://www.collegenet.com/r25" xmlns:xl="http://www.w3.org/1999/xlink" pubdate
="2021-06-15T14:26:43-07:00" engine="sws">
   <r25:login>
      <r25:message>Login successful</r25:message>
      <r25:success>T</r25:success>
      <r25:user_type>r25</r25:user_type>
      <r25:user_id>3143</r25:user_id>
      <r25:username>25livedemo</r25:username>
      <r25:contact_name>Demo, 25live</r25:contact_name>
      <r25:security_group_id>2</r25:security_group_id>
      <r25:security_group_name>Academics - Advanced (2)</r25:security_group_name>
      <r25:login_url></r25:login_url>
      <r25:logout_url>https://webservices.collegenet.com/r25ws/wrd/<instance>/run/logout.xml</r25:logout_url>
   </r25:login>
</r25:login_response>
```

## Digest Authentication

Digest can be handled automatically by the browser . This type of auth can also be used by scripts/code to negotiate authentication. When using digest, there is a sequence of communication between the browser and the server, where the server creates a hash that the browser must respond with (all happens in a single "request").

1. A client requests access to a protected resource.
2. The web server returns a dialog box that requests the user name and password.
3. The client submits the user name and password to the server.
4. The server authenticates the user in the specified realm and if successful, returns the requested resource.
   a. The **authentication** parameter '**realm**' is **REQUIRED and must be supplied in the response back TO the server.**
   b. The HTTP **`WWW-Authenticate`** response header defines the authentication method that should be used to

gain access to a resource.

The server responds with:

**Here's an example of the authorization header on a request after digest authorization has taken place:**

```
HEADER
Authorization: Digest username="25livedemo", realm="R25R25 WebServices Web Services", nonce="fd2031312e8c
8285a0aaf47d29e48", uri="/r25/servlet/wrd/run/login.xml", algorithm=MD5,
  response="e4ba450ac6e868499cd72b231972253c", qop="auth", nc=00000001,
  cnonce="682d7970a88181deed251a1fded072ea"

HTTP
GET /r25ws/wrd/<instance>/run/login.xml HTTP/1.1
Host: webservices.collegenet.com
Authorization: Digest username="25livedemo", realm="R25 WebServices", nonce="MTYyMzk0NDA2MTkzMTo0NjQyM
TIzMzBiYjJkNTVkZGJhMTdkZDViZGRjMWM4NQ==", uri="/25live/data/<instance>/run/login.xml", response="1f7862cc
00b65940025d0ec18837f4fd", qop=auth, nc=00000002, cnonce="30be57b0c1ef5047"
Cookie: Blaze=YNJp0aDPxw-u815AjWF2vgAALNM; WSSESSIONID=0EAFDA8B3070F8F1DDB2FAB7753BC6C2; BIGipSer
verp-java.webservices-web.collegenet.com=2249459466.36895.0000
```

The server will generate a unique nonce value for each session.

The client generates a response string to prove that they know a password. The server will be able to compare this string against a version it can generate from the assumed same information.

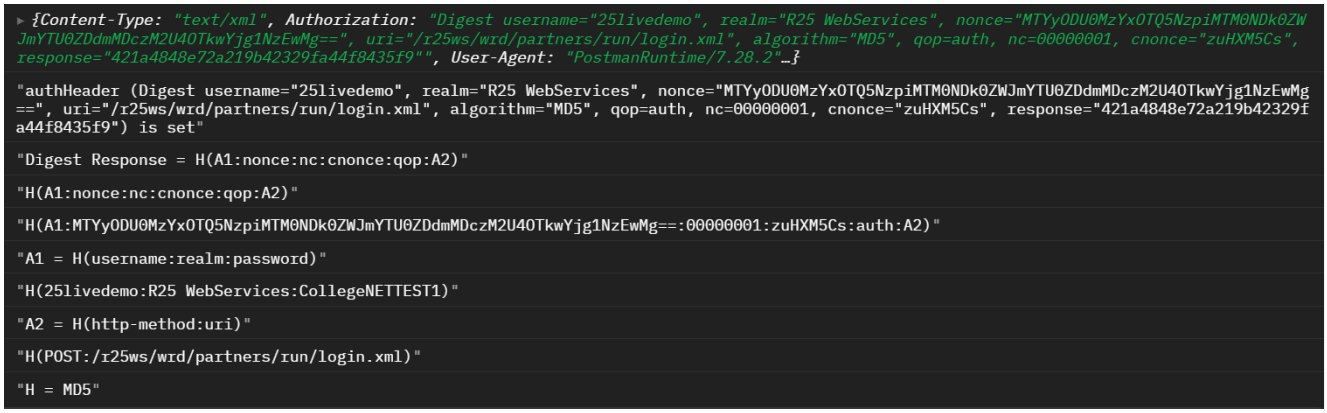
**Response** = **H**(**A1**:nonce:nc:cnonce:qop:**A2**)
**A1** = **H**(username:realm:password)
**A2** = **H**(http-method:uri)

## Parameter Translation

- **H** = **MD5**
- **username** = R25 username
- **realm** = Realm supplied by Web Server (R25 WebServices)
- **password** = R25 password
- **http-method** = One of GET, POST, PUT or DELETE
- **uri** = Full path to the requested resource
- **nonce** = Nonce value from the server
- **nc** = Nonce count, or number of requests to the server (in hex)
- **cnonce** = Client generated nonce
- **qop** = Quality of protection (auth in R25 WebServices)

It might be helpful to use a tool such as POSTMAN  and a custom script to capture the parameter variables being used on the GET Call to run/login.xml so that it's easier to populate the **POST** to login.xml, after that.

Example:




## Cleaning up your session

After your session has been marked as valid, you will not need to provide credentials again. To cleanup your session, just make a **GET** request to **logout.xml**

**GET** *https://webservices.collegenet.com/r25ws/wrd/<instance>/run/logout.xml*

**logout.xml GET response**

```
<?xml version="1.0" encoding="utf-8"?>
<r25:goodbye xmlns:r25="http://www.collegenet.com/r25" xmlns:xl="http://www.w3.org/1999/xlink" engine="sws">
25livedemo</r25:goodbye>
```