How Does HTTP Authentication Work?

We all use cookies to create a authentication system. But most of us don't know that HTTP provides a authentication model which can be implemented within minutes by writing little code. In this post we will look at creating a authentication system using HTTP's built in mechanism.

What is HTTP Authentication?

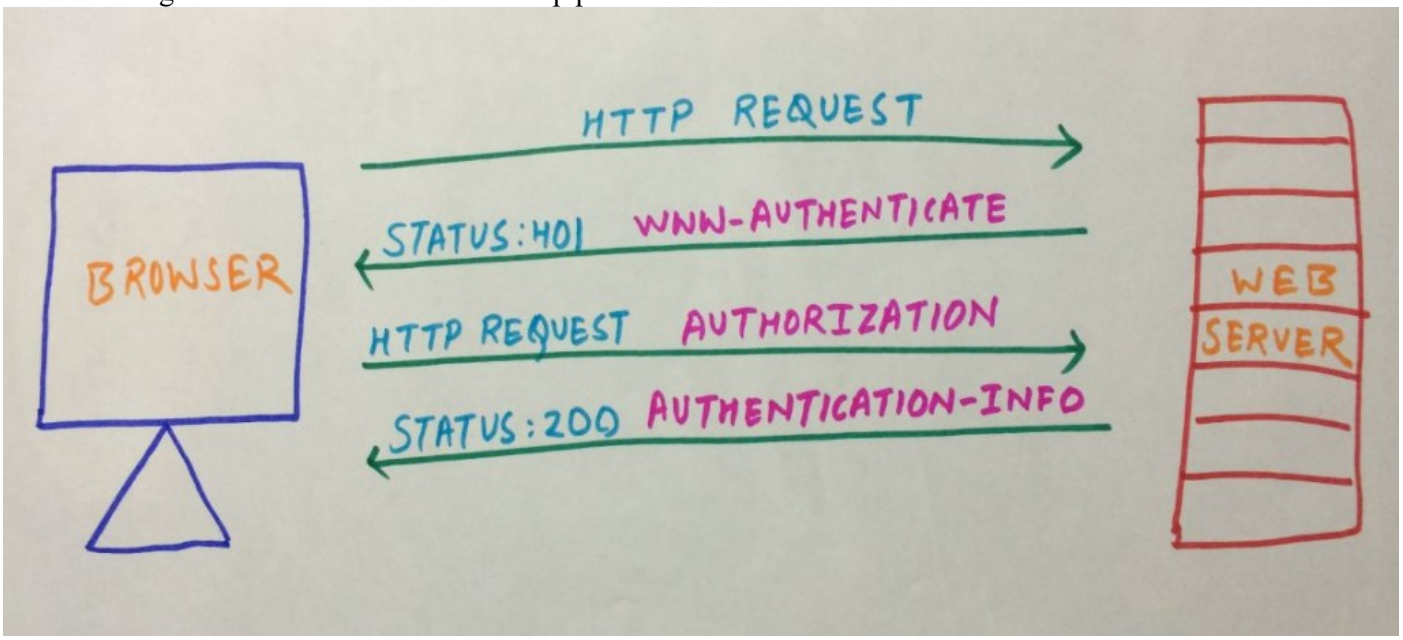
Authenticating users using HTTP's built in authenticating mechanism.

Types of HTTP Authentication

HTTP provides two ways to authenticate users i.e., Basic authentication and Digest authentication.

HTTP Authentication Phases

Basic and Digest authentication use a four step process to authenticate users.



First HTTP client makes a request to the web server. Request method doesn't has to be GET it can be any method. If web server sees that the requested resource need authentication to access then it sends backs 401 Unauthorized status code along with WWW-Authenticate header. And then client displays a dialog box to take username and password as input. Once the credentials has been enter the client sends it using the Authorization header. If the credentials are correct then server responds with 200 status code and Authentication-Info header.

If client sends wrong credentials in the Authorization request then server again responds with 401 status code. The client is allowed to try again and again.

This is the basic process followed by Basic and Digest models. Values assigned to the authentication headers is different for both models, this is why they differ.

Basic Authentication

Let's look at the authentication headers in depth for Basic authentication.

WWW-Authenticate -> This header is assigned to a realm. It is compulsory that this header will contain a realm directive. realm is displayed in the dialog box. Servers use realm to group different parts of the server(assigns same realm, username and password other resources on the same and deeper level). Browser saves credentials for all realm's. Whenever browser receives a WWW-Authenticate response with a realm already saved, it will automatically send the credentials without the knowledge of user. Browser also sends Authorization request directly to the URLs deeper than a level whose realm and credentials are known. This creates a session among the URls with same realm and also URIs deeper to an saved realm.

WWW-Authenticate: realm="Videos"

Authorization -> Browser sends the username and password using this header. username and password is joined together with a colon in between and then encoded using base-64 encoding method. So if the username is "narayanprusty" and the password is "qnimate" than a string "narayan:prusty" is generated and then encoded using base-64 that results to the string "D08mRvgvbhDsU". And this final string is sended to the server.

Authorization: Basic D08mRvgvbhDsU

Authentication-Info -> This header is optional. Some web servers send this header assigned to some information about the session and future authentication requests.

Some organizations use proxy servers to authenticate users before allowing them to access the web server resources. The same mechanism is used by proxies to authenticate users but header and status codes are changed. Changes are:

- Response status 407: 407 response status code is sended instead of 401
- Proxy-Authenticate: Proxy-Authenticate is used instead of WWW-Authenticate.
- Proxy-Authorization: Proxy-Authorization is used instead of Authorization.
- Proxy-Authentication-Info: Proxy-Authentication-Info is used instead of Authentication-Info.

Digest Authentication

Let's look at authentication headers in depth for digest authentication.

WWW-Authenticate -> This header is assigned to realm, qop, nonce, stale, opaque, domain and algorithm directives. Let's see values of each directive.

- realm: realm works the same way as it works for basic authentication.
- qop: qop stands for quality of protection. Its a optional directive. It can have these values "auth", "auth-int" or "auth,auth-int". This directive prevents from replay attacks and choosen plaintext attacks.
- nonce: A unique value generated for every 401 response. Its calculated using time and some other values. This value has a expiry time and has a limit of number of times it can be used. This helps to prevent replay attacks. To learn more about replay attack click here

- stale: Its a optional directive. Its set to true if client has used a nonce which is invalid while sending a authorization request. And then client is allowed to retry with the new nonce provided.
- domain: Its a optional directive. A comma-separated list of URIs is assigned to the directive. This list indicates that all the list URLs have the same credentials as the request URL. So now browser saves this list and credentials. While making a request to a URL from the list the browser directly requests with the Authorization header instead of making a normal request and then matching the realm or assuming a realm. One more important thing is that the list can be cross domain.
    - If this directive is not provided then the browser assumes that all the URL's in the domain have the same credentials. So therefore while making request to any URL after the first URL request, the client directly makes an request with Authorization header. If a URL has two credentials due to domain directive overriding then the latest credential is used.
- opaque: Its a optional directive. Its a string of data, specified by the server, which should be returned by the client unchanged. It is recommended that this string be base64 or hexadecimal data. The opaque data is useful for transporting state information around. For example, a server could be responsible for authenticating content which actually sits on another server. The first 401 response would include a domain field which includes the URI on the second server, and the opaque field for specifying state information. The client will retry the request, at which time the server may respond with a 301/302 redirection, pointing to the URI on the second server. The client will follow the redirection, and pass the same Authorization header, including the opaque value which the second server may require.
- algorithm: Its a optional directive. It can have these values "MD5″ or "MD5-sess"

```
WWW-Authenticate: Digest realm="Videos", qop="auth,auth-int", nonce="jd839ud9832duj329u9u8ru32rr8u293ur9u32
```

Authorization -> This header is assigned to username, realm, nonce, uri, qop, nc, cnonce and response directives. Let's see the values of each directive.

- username: username entered by the user in the dialog box.
- realm: same realm as provided by the server.
- nonce: same nonce provided by the server.
- uri: uri of the resource which needs to be accessed.
- qop: In case server sends us qop values as "auth,auth-int" then we need to choose any one among the two. So the server also needs to know the qop we selected. We provide the selected value using this qop directive.
- nc: This directive MUST be specified if a qop directive is sent (see above), and MUST NOT be specified if the server did not send a qop directive in the WWW-Authenticate header field. The value is the hexadecimal count of the number of requests (including the current request) that the client has sent with the nonce value in this request. For example, in the first request sent in response to a given nonce value, the client sends "nc=00000001″. The purpose of this directive is to allow the server to detect request replays by maintaining its own copy of this count – if the same value is seen twice, then the request is a replay.
- cnonce: Its a unique value generated by the client. This prevents from chosen plaintext attacks. This MUST be specified if a qop directive is sent (see above), and MUST NOT be specified if the server did not send a qop directive in the WWW-Authenticate header field.
- response: Credentials are hashed and assigned to this directive. The way credentials are hashed depends on the qop and algoritm directives.

If the algorithm directive's value is "MD5" or unspecified, then HA1 is

$$\mathrm{HA1} = \mathrm{MD5}\Big(\mathrm{A1}\Big) = \mathrm{MD5}\Big(\mathrm{username} : \mathrm{realm} : \mathrm{password}\Big)$$

If the algorithm directive's value is "MD5-sess", then HA1 is

$$\mathrm{HA1} = \mathrm{MD5}\Big(\mathrm{A1}\Big) = \mathrm{MD5}\Big(\mathrm{MD5}\Big(\mathrm{username} : \mathrm{realm} : \mathrm{password}\Big) : \mathrm{nonce} : \mathrm{cnonce}\Big)$$

If the qop directive's value is "auth" or is unspecified, then HA2 is

$$\mathrm{HA2} = \mathrm{MD5}\Big(\mathrm{A2}\Big) = \mathrm{MD5}\Big(\mathrm{method} : \mathrm{digestURI}\Big)$$

If the qop directive's value is "auth-int", then HA2 is

$$\mathrm{HA2} = \mathrm{MD5}\Big(\mathrm{A2}\Big) = \mathrm{MD5}\Big(\mathrm{method} : \mathrm{digestURI} : \mathrm{MD5}(entityBody)\Big)$$

If the qop directive's value is "auth" or "auth-int", then compute the response as follows:

$$\mathrm{response} = \mathrm{MD5}\Big(\mathrm{HA1} : \mathrm{nonce} : \mathrm{nonceCount} : \mathrm{clientNonce} : \mathrm{qop} : \mathrm{HA2}\Big)$$

If the qop directive is unspecified, then compute the response as follows:

$$\mathrm{response} = \mathrm{MD5}\Big(\mathrm{HA1} : \mathrm{nonce} : \mathrm{HA2}\Big)$$

Authorization: Digest username="narayanprusty", realm="Videos", nonce="jd839ud9832duj329u9u8ru32rr8u293ur9u3

Authentication-Info -> This header is sended by the server if the authentication is successful. This header can be assigned to many different values according to the way server and client are designed. Most of the times this header is used to pass information to the client about the next authentication request. One popular directive this header is assigned to is nextnonce. nextnonce is assigned to a nonce will be used by the client to make an Authorization request directly instead of making a normal request and receiving the nonce. This decreases the number of requests made.

Preemptive Authentication

Preemptive authentication is when browser has the confidence to skip 1st and 2nd phase and jump to the 3rd phase directly.

In basic authentication clients saves credentials for every URL and realm so that it can be a preemptive authentication.

In digest authentication clients make use of domain directive, nextnonce directive, saved credentials and saved realm to make it a preemptive authentication.

Setting HTTP authentication using .htacces

For setting up HTTP authentication we can use any web server or we can manually write server side scripts for HTTP authentication. Here I assume you are using Apache HTTP web server.

For setting up basic authentication for root web directory place these two files in the root web directory
.htaccess

```
AuthType Basic
AuthName "realm-name"
AuthUserFile system-path/to/.htpasswd/file
AuthUser valid-user
```

.htpasswd

#username is narayanprusty and password is qnimate. Here password is base64 encoded.
narayanprusty:6AcmcO7GMICnU

Now all files inside the web directory will have basic authentication protection with the same realm and credentials.

For setting up digest authentication for root web directory place these two files in the root web directory
.htaccess

AuthType Digest
AuthName "realm-name"
AuthUserFile system-path/to/.htpasswd/file
Require valid-user

.htpasswd

#here password is also qnimate. The hash is MD5(narayanprusty:realm-name:qnimate) = HA1
narayanprusty:realm-name:a2ec4f5136d7fadac1c948e2400eb703

For more information on authentication using .htaccess visit here

Logging Out User

To logout user from a session, web server needs to respond with 401 status code for Authorization request.

Conclusion

Digest authentication is more secure than basic authentication. Modern hacking tools can easily break digest authentication. So its always better to cover these models with TLS. I put a lot of explanation on how realm is used to create session among URLs and also how domain directive in digest can be used to make credentials of two URls same. Thanks for reading this.

May 3, 2014Narayan Prusty
How Web Caching Works?Storing Meta Data In HTML Tags Using data-* Attributes

Leave a Reply

-
-
-

Type here

Post Comment