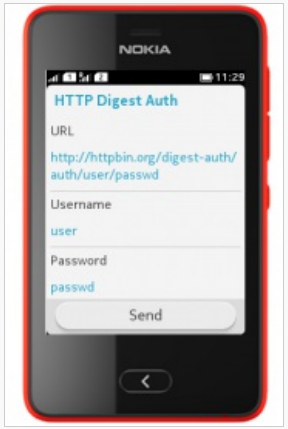
HTTP digest access authentication in Java ME

This article explains how to implement HTTP digest access authentication in Java ME.

Introduction

Digest access is an authentication method used to protect HTTP resources via a negotiation-based authorization mechanism. Compared to basic access authentication, it offers a higher level of protection, avoiding the need to send authentication data in clear text.



Java ME app using digest access authentication

The typical flow of digest access authentication is:

1. the client sends an unauthorized HTTP request to the server, without specifying any credentials
2. the server replies with a 401 Unauthorized HTTP response code, specifying a *WWW-Authenticate* HTTP header containing relevant information to start the authentication process
3. the client parses the *WWW-Authenticate* header, computes and sends back to the server a new HTTP request with a *Authorization* HTTP header containing the calculated authentication data
4. if the authentication completes successfully, the server replies with a *200 HTTP response code*

This article illustrates a possible implementation of digest access authentication in Java ME, building the functionality on the networking features and APIs offered by the platform.

Implementation

The following sections describe the various steps of a the digest access authentication implementation, and specifically:

- reading the *WWW-Authenticate* response header
- parsing the values provided within the *WWW-Authenticate* header
- computing the response MD5 hash value
- generating the *Authorization* HTTP header
- sending an authenticated request using the computed *Authorization* header

The unauthenticated request

The Java app sends a plain HTTP request, without specifying any credentials and, if digest access authentication is used, receives a *401 Unauthorized* response code. In this case, the app checks if the content of the *WWW-Authenticate* header defines the "Digest" access authentication method, before proceeding with further steps.

```
HttpConnectionhc = (HttpConnection) Connector.open(url);

int responseCode = hc.getResponseCode();

if(responseCode == HttpConnection.HTTP_UNAUTHORIZED)
{
 String wwwAuthHeader = (String)hc.getHeaderField("WWW-Authenticate");
```


(C) Copyright Nokia 2014. All rights reserved.

```
 if(wwwAuthHeader != null && wwwAuthHeader.indexOf("Digest ") == 0)
 {
  // digest access authentication is used
 }
}
```

The WWW-Authenticate header

The *WWW-Authenticate* HTTP header, sent by the server within replies to unauthorized HTTP requests, contains all the pieces needed by the client to compute, starting from the user credentials, the correct *Authorization* header that must be used to access the specified HTTP resource.

Specifically, the *WWW-Authenticate* header is built by the following pieces:

- *realm*: identifies the realm the user should authenticate to
- *qop*: identifies the *quality of protection* - can assume different values, but the most widely used is *auth*
- *nonce*: a randomly generated unique string
- *opaque*: another random string

A sample *WWW-Authenticate* header is the following:

```
WWW-Authenticate: Digest realm="My Realm",
  qop="auth",
  nonce="32934ae8349d",
  opaque="348734dec9237af3388"
```

Note: This article shows how to implement digest access authentication when *quality of protection*'s value is *auth*. Other variants can be implemented using this tutorial as a starting point.

The following code snippet parses the *WWW-Authenticate* header, removing value quotes where used, and putting all key/value pairs into a `Hashtable`.

```
Hashtable parseHttpDigest(String parts)
{
    Hashtable data = new Hashtable();

    int equalIndex;

    while((equalIndex = parts.indexOf("=")) >= 0)
 {
     String partName = parts.substring(0, equalIndex).trim();

     int endIndex = parts.indexOf(",", equalIndex);
     if(endIndex == -1)
      endIndex = parts.length();

     String partValue = parts.substring(equalIndex + 1, endIndex).trim();

     if(partValue.charAt(0) == '"')
      partValue = partValue.substring(1, partValue.length() - 1);

     data.put(partName, partValue);

     if(endIndex == parts.length())
      break;
```


(C) Copyright Nokia 2014. All rights reserved.

```
        parts = parts.substring(endIndex + 1);
    }

    return data;
}
```

Computing the response hash

Using the values parsed from the *WWW-Authenticate* HTTP header, the app must compute a response MD5 hash. Since Java ME has no inbuilt support for MD5, a custom solution must be used, as the one available here: MD5 hash in Java ME.

The response hash in computed in the following steps:

- A first MD5 hash is computed by using the supplied username and password, together with the realm specified by the server:

```
String A1 = md5(user + ":" + realm + ":" + pass);
```

- A second MD5 hash is computed using the HTTP request method and the HTTP resource URI:

```
String A2 = md5(httpMethod + ":" + uri);
```

- Before computing the final MD5 hash, the client must generate a random string, and increment (or initialize) a hex counter:

```
String cnonce = Integer.toString(Math.abs(new Random().nextInt()));
String ncvalue = "00000001";
```

- By using the above values, together with the *nonce* and *qop* values supplied by the server, the final hash is computed as follows:

```
String responseSeed = A1 + ":" + nonce + ":" + ncvalue + ":" + cnonce + ":" + qop + ":"
+ A2;
String response = md5(responseSeed);
```

The Authorization header

Once the response hash is calculated, the app can build the *Authorization* HTTP header that must be used to send the authenticated HTTP request. The *Authorization* header must be prefixed by the "*Digest*" string, and is built up of the following pieces:

- *username*: the username supplied by the user
- *realm*: same value sent by the server
- *nonce*: same value sent by the server
- *uri*: the URI that the client is accessing
- *opaque*: same value sent by the server
- *qop*: same value sent by the server
- *nc*: a hex number that must be increased by the client for each request
- *cnonce*: a random string generated by the client
- *response*: the hash computed by the client starting from the *WWW-Authenticate* header sent by the server

An *Authorization* header can be built in Java ME as follows:

```
String authorizationHeader  = "Digest username=\"" + user + "\", realm=\"";

authorizationHeader += realm + "\", nonce=\"" + nonce + "\",";

authorizationHeader += " uri=\"" + uri + "\", cnonce=\"" + cnonce;
```


(C) Copyright Nokia 2014. All rights reserved.

```
authorizationHeader += "\", nc=" +ncvalue + ", response=\"" + response + "\", qop=" +
qop;

authorizationHeader += ", opaque=\"" + opaque + "\"";
```

The authenticated request

At this point, an authenticated HTTP request can be built by setting the *Authorization* header with the value computed in the previous step and, if the used credentials are correct, the HTTP server replies with a *200 response code*.

```
HttpConnection hc = (HttpConnection) Connector.open(url);

hc.setRequestProperty("Authorization", authorizationHeader);

responseCode = hc.getResponseCode();
```

The server replies to the authenticated request with a 200 response code

Testing

HTTPBin.org provides a useful service that can be used to test digest access authentication: using the URL http://httpbin.org/digest-auth/{USERNAME}/{PASSWORD} , where {USERNAME} and {PASSWORD} can be customized with the desired user credentials, Java app can be tested without a production server available.

The sample Java app attached to this article uses that testing server, but data can be easily changed in order to test against other testing or production servers.

Summary

This article illustrates a possible Java ME implementation of HTTP digest access authentication.

Full source code of the sample app illustrated in this article is available here: Media:WikiHttpDigestAuth.zip.

(C) Copyright Nokia 2014. All rights reserved.

(C) Copyright Nokia 2014. All rights reserved.