Tecnologie e applicazioni web

Authentication

Filippo Bergamasco (filippo.bergamasco@unive.it)
http://www.dais.unive.it/~bergamasco/
DAIS - Università Ca'Foscari di Venezia
Academic year: 2021/2022

Authentication

Cookies allows a web server to store key-value pars to the user agent requesting a certain resource

With this technique a we can recognize the same client throughout subsequent requests but not its “real identity”

HTTP support various mechanisms to **authenticate** a client by providing its own credentials

Authentication

Authentication means **showing some evidence of the true physical identity of a certain client**

It is usually based on some **shared information** between client and server (ex. username-password pair) that must be exchanged securely with a pre-defined protocol

HTTP Authentication

When a client requests a protected resource, the server may respond with the status code **401** (login required)

Together with the status code, the

`WWW-authenticate` header informs the user agent to the kind of data that must be provided to authenticate

User login

The web browser, according to the kind of authentication requested:

- Ask the user for a user/password pair
- Creates an HTTP header containing the login credentials
- Uses the header for all the subsequent resources under a certain “authorization realm”

Similar to the cookie mechanism: authentication credentials are exchanged in HTTP **headers**.

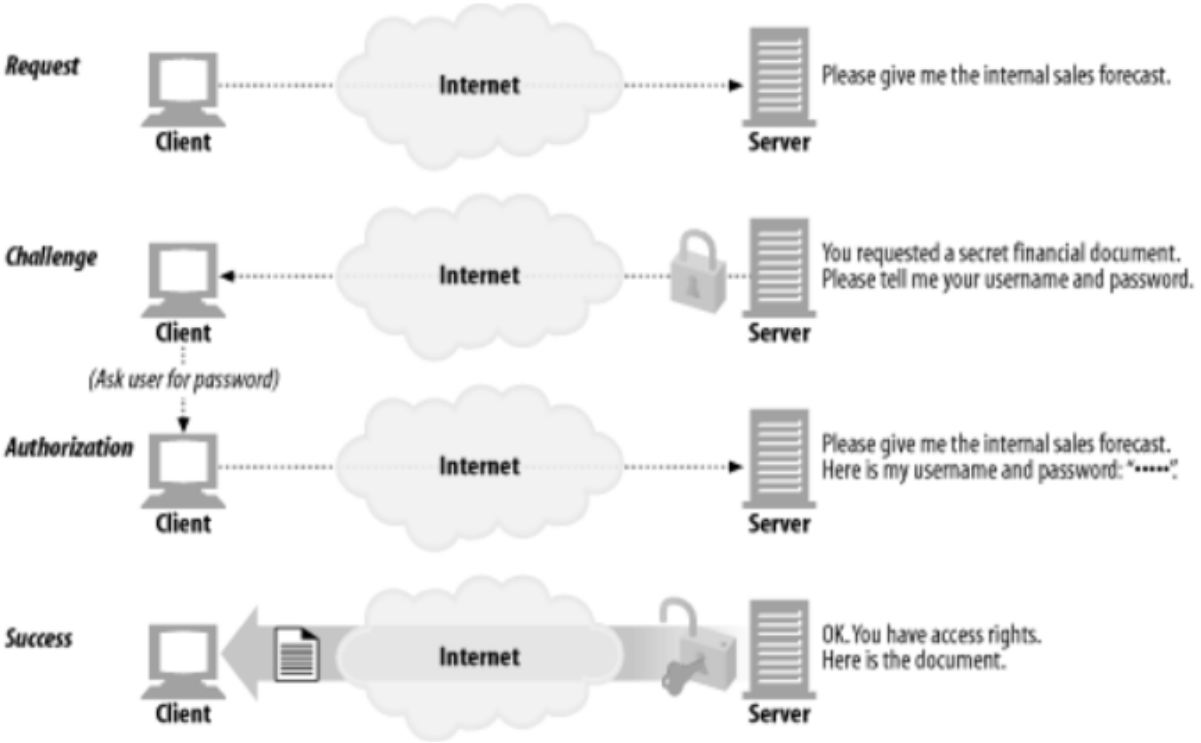
Authentication types

HTTP implements 2 authentication mechanisms:

1. Basic access authentication
2. Digest access authentication

Both based on a challenge-response framework, but differ in the way in which the information is encoded and exchanged.

Challenge-response



< Challenge

< Response
(sent in the request following the challenge)

Basic authentication

Originally described in HTTP/1.0, nowadays implemented in all web browsers.

- A server may reject a transaction, challenging the client to send a `username:password` pair.
- If the reply is correct, the resource is sent in the next transaction
- If the reply is not correct, the resource is not sent and the challenge is repeated

Basic authentication

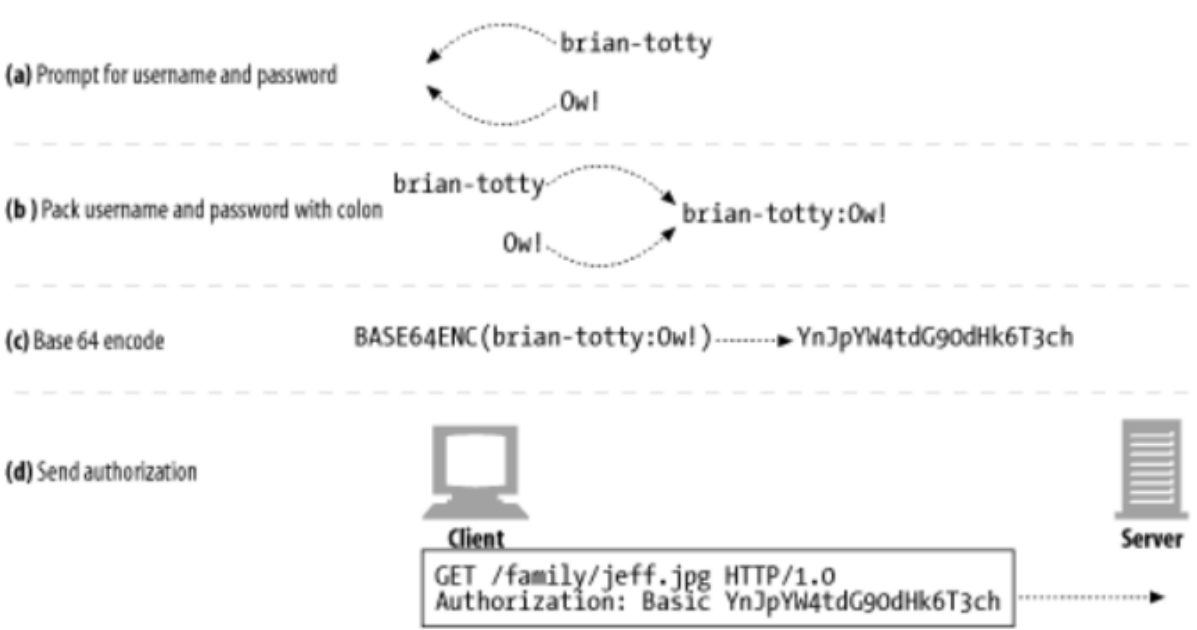
1. Client sends a **request** (GET, POST, HEAD, etc.) to access a certain resource
2. If the resource is protected, the server insert the following header in the **response**:

   `WWW-Authenticate: Basic realm="<realm-name>"`

3. Client asks username and password to the user and encode the data as a string:

   <crd> = `base64(<username>:<password>)`

Base-64 encoding



Basic authentication

4. The user agent, for every subsequent request, inserts the header

```
Authorization: Basic <crd>
```

The server decodes <crd> in base64 to obtain the <username>:<password> string and verifies the credentials. If valid, the resource is provided as happens without authentication.

Base-64 encoding

Base64 is used to encode a generic byte stream to a string containing alphanumeric characters only.

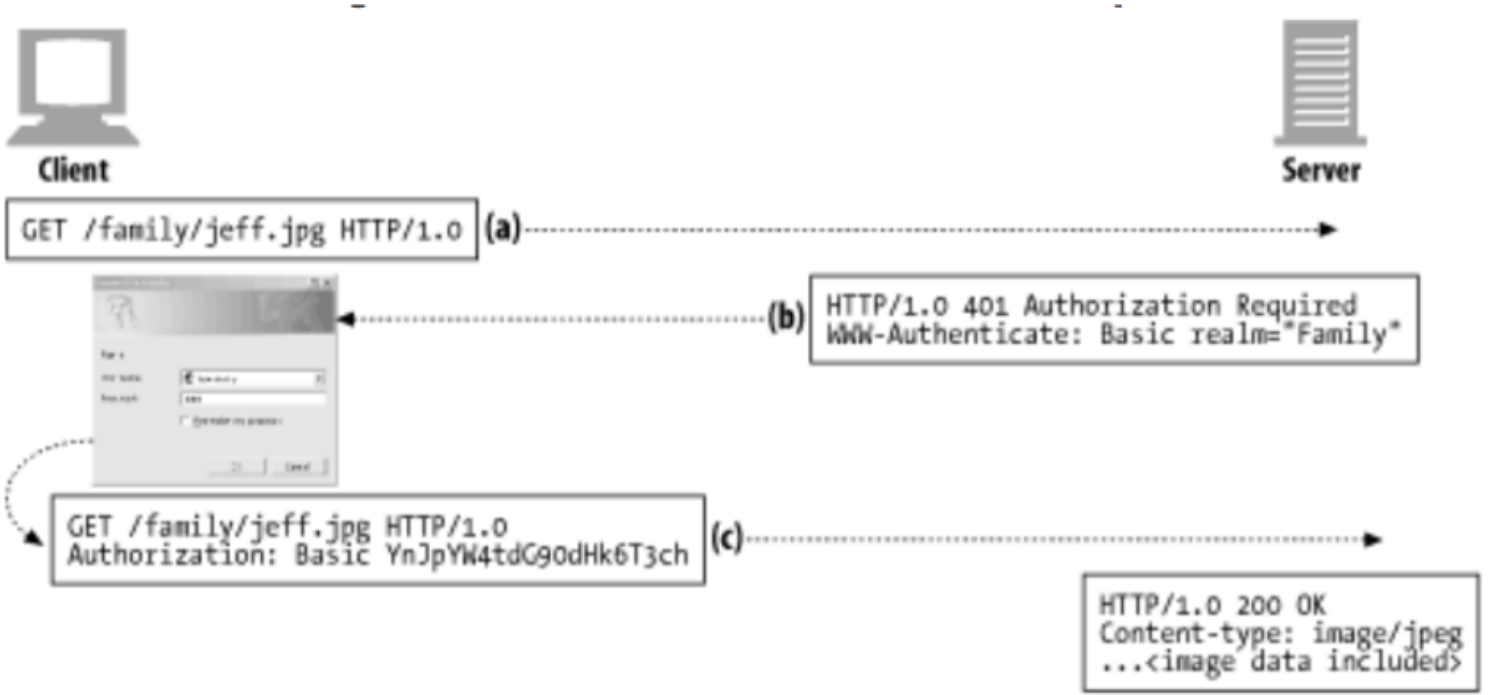
Important: Base64 is not meant to encrypt the data, that remains easily readable to anyone. The user:password pair is just obfuscated (not secure at all)

Base-64 encoding

Why not just sending <username>:<password> with no encoding at all?

1. Non non plain-ascii characters in the password can be safely inserted in the HTTP headers
2. String is obfuscated to avoid humans to simply read the password if the HTTP traffic is observed

Basic authentication



BA: problems

1. User credentials can be are easily **decoded** by anyone. Same security level of sending user/password with no encryption at all.

Solution: Use basic authentication with HTTPS only

BA: problems

2. Even if the authentication is used for non-critical applications, users may still recycle passwords used for other websites or sensible applications

Problem: Social behavior of the users

BA: problems

3. Even if the authentication happens correctly, the provided resource is not necessarily bound to the provided credentials.

A man-in-the-middle can change the resource data without tampering the authentication headers

Solution: Again, basic authentication should only be used with HTTPS

BA: problems

4. **Server spoofing**: client cannot verify the server true identity that can therefore be impersonated by a malicious entity.

Solution: Use digital certificates to authenticate the server before using basic access authentication (again, HTTPS solves this problem)

BA: Usages

If not coupled with HTTPS, BA is only useful to give a naïve authentication layer where content privacy is desired but not strictly necessary.

Just blocks “curious” users but offers no real security guarantees…

Digest Authentication

Basic Authentication is insecure because the username/password pair is sent without encryption.

But.. symmetric/asymmetric key techniques to create a secure channel tend to overcomplicate the protocol

Digest Authentication

Rationale:

A server does not necessarily needs to receive the password (shared secret) from the client. A **digest** is sufficient to prove that a client actually knows the correct password

Hashing functions are commonly used to compute those digest

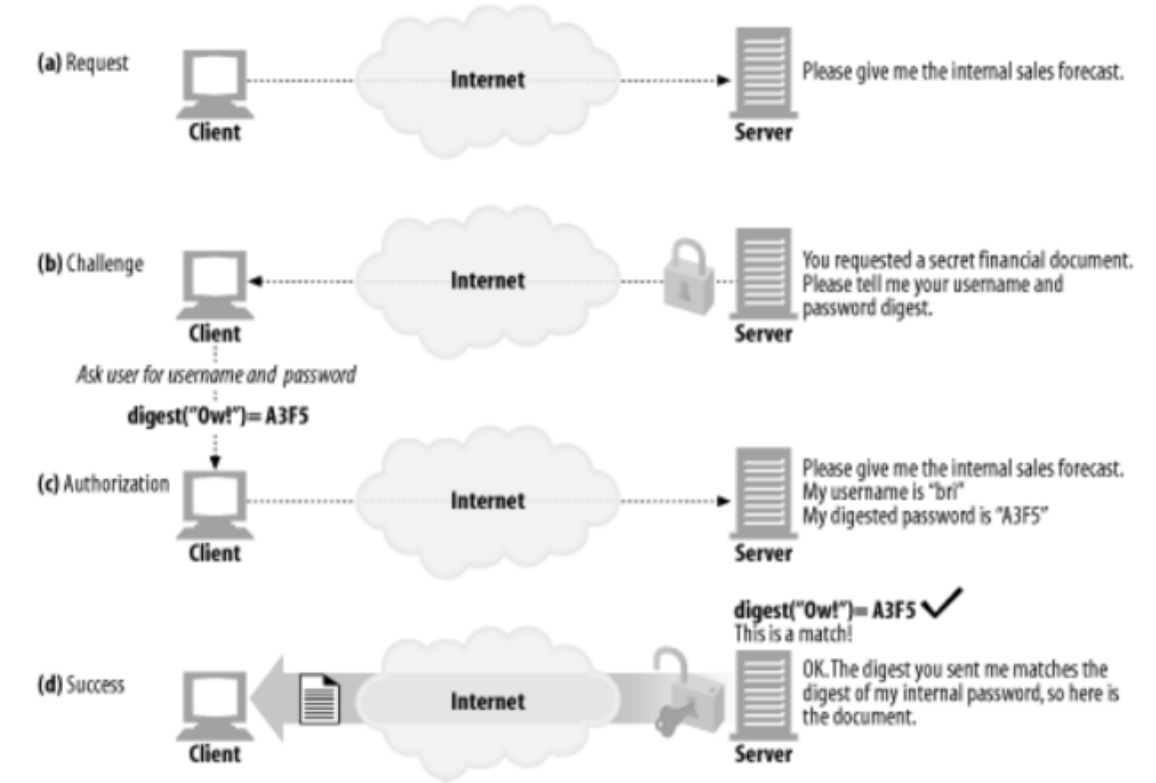
Hash function

A hash function can convert any message (string) to a fixed-length sequence of (random-like) bytes

Features:

- Equal messages generate equal hashes
- **Hash function is one-way**: practically impossible to recover the original message from its hash
- Collisions are possible, but extremely rare

Digest authentication



Digest authentication

With a hash function we can avoid sending the password directly!

Problem: Since a password always generates the same digest, anyone eavesdropping the channel can steal the digest and use it to authenticate without knowing the password: **Replay attack**

Nonce

To avoid the replay-attack, the server send a special token (called **nonce**) as part of the authentication challenge

The client computes hash(<password>,<nonce>) so that each digest is different and usable only once (supposing that nonces change every time)

Digest Authentication

The (simplified) digest authentication protocol works as follows:

1. Similar to BA, the server sends the `WWW-Authenticate` header in its response, specifying the authentication realm and a randomly generated nonce

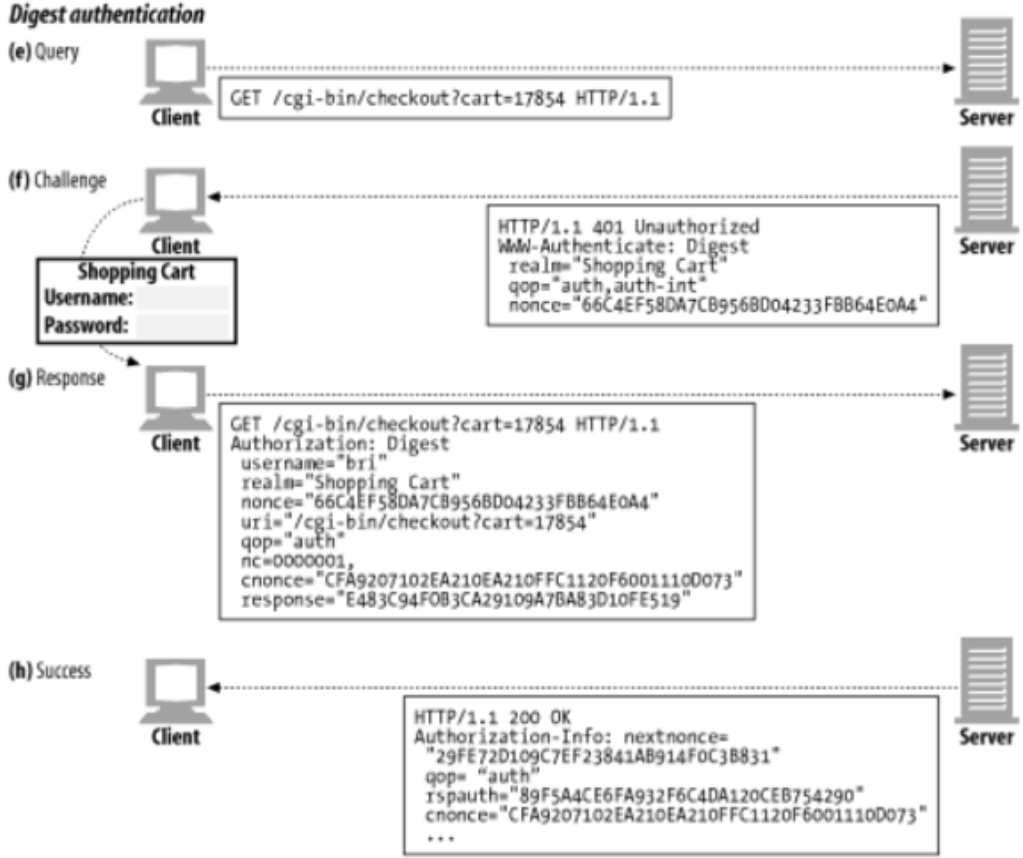
Digest Authentication

2. The client creates the digest of the triple (user, nonce, password)
3. The client sends in the Authorization header its own username and the digest.
4. The server creates its own digest (with user, nonce and password) to check if it matches with the one sent by the client

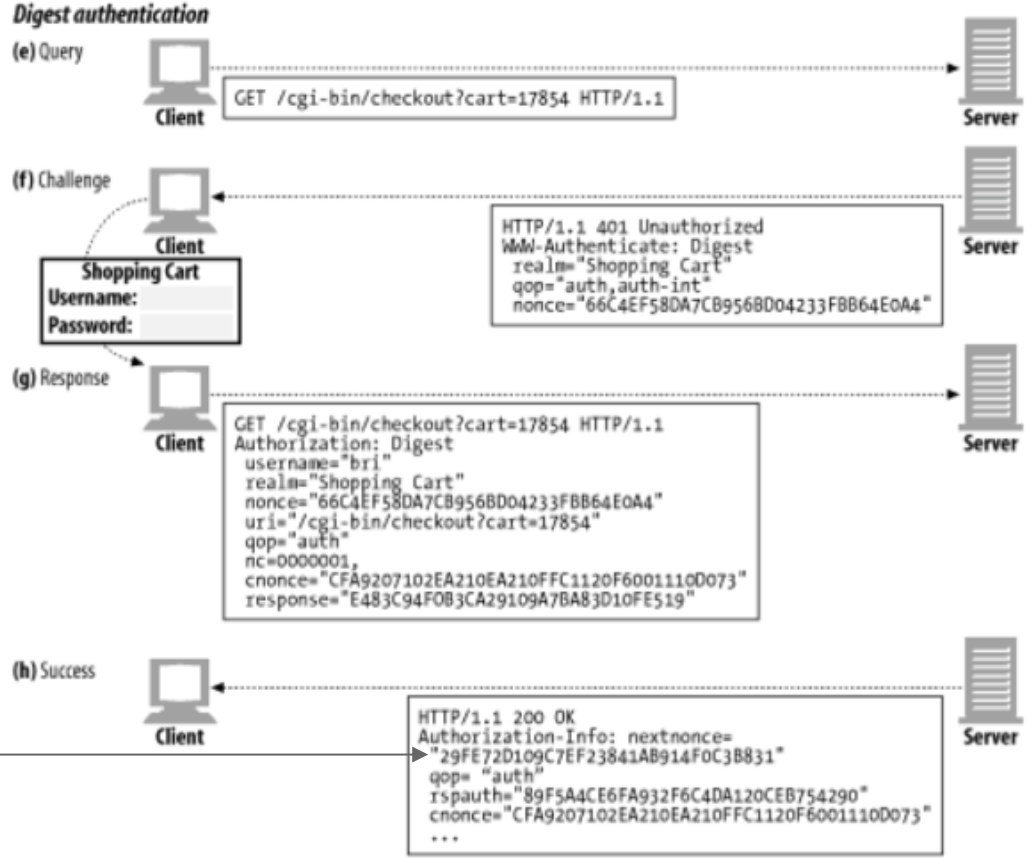
Digest Authentication

5. If the two digests matches, the client is authenticated. If not, the request is refused and a new challenge (ie. a new nonce) is generated
6. If the client had in turn sent a nonce, the digest for the client is generated and returned in the Authorization-Info header

Digest Authentication

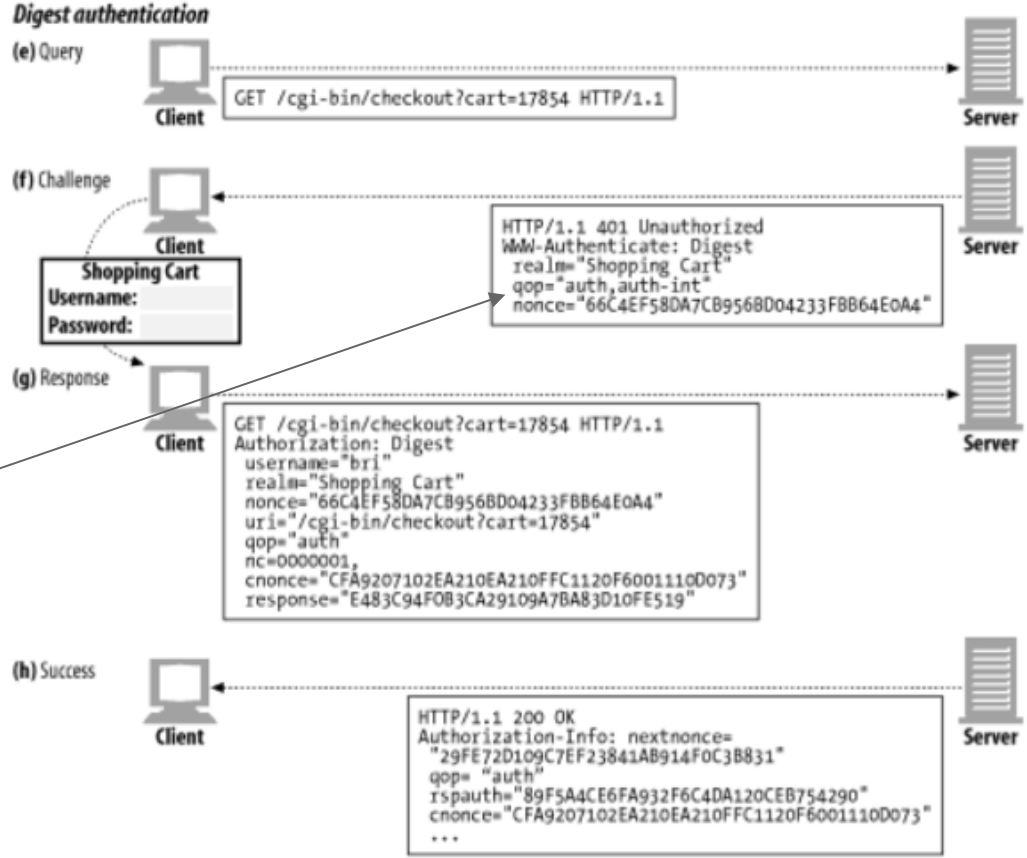


Digest Authentication



nextnonce allows the client to send a new request with the Authorization header already present, skipping a new challenge initiated by the server

Digest Authentication



Qop (quality of protection enhancements) allows client-server negotiation on the security features to use (ex. Integrity protection also for message bodies)

DA: Advantages

- Password is not sent in cleartext anymore!
- Replay attack not possible (at least if the nonce is randomly generated every time)
- Client can verify if the server is the same entity that generated the challenge (client nonce)

DA: Problems

- Different security profiles (for retro-compatibility with legacy version) may lead to insecure implementations
- Server true identity cannot be verified! Indeed, we can only verify that the server is the same entity that generated the challenge (man-in-the-middle attack still possible)
- MD5 algorithm considered insecure nowadays