Basic Authentication

- It's a Stateless Authentication method.
  - Stateless authentication means, server do not maintains the user authentication state (aka Session).
- In this, client has to pass the username and password with every request using Authorization header
  Authorization: Basic <base64(username:password)>
- These Credentials are encoded using Base64 (not encrypted), making it insecure over HTTP.

Let's go step by step:

1st: Lets create a user

- Already discussed, all possible ways to create user and how we can create users dynamically and with industry standard approach.
- Also how to create and store it in DB or in-memory. So kindly check that out if there is any doubt with user creation process.

Lets create a user for testing purpose:

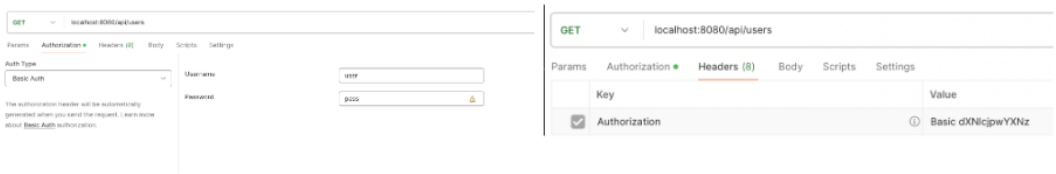
Application.properties

```
#creating username and password and assigning the ROLE to the user
spring.security.user.name=user
spring.security.user.password=pass
spring.security.user.roles=ADMIN
```

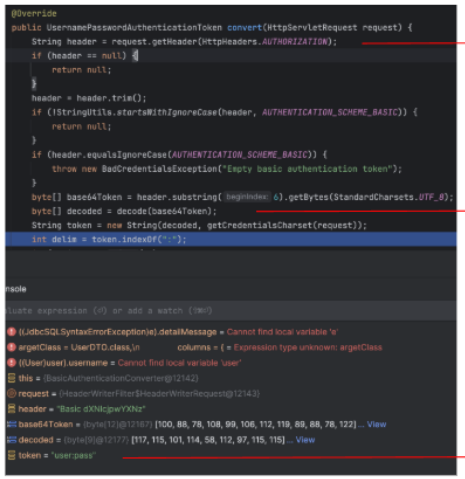
Lets create a user for testing purpose:

Application.properties

```
#creating username and password and assigning the ROLE to the user
spring.security.user.name=user
spring.security.user.password=pass
spring.security.user.roles=ADMIN
```



BasicAuthenticationFilter.java



```
@Override
public UsernamePasswordAuthenticationToken convert(HttpServletRequest request) {
    String header = request.getHeader(HttpHeaders.AUTHORIZATION);
    if (header == null) {
        return null;
    }
    header = header.trim();
    if (!StringUtils.startsWithIgnoreCase(header, AUTHENTICATION_SCHEME_BASIC)) {
        return null;
    }
    if (header.equalsIgnoreCase(AUTHENTICATION_SCHEME_BASIC)) {
        throw new BadCredentialsException("Empty basic authentication token");
    }
    byte[] base64Token = header.substring( beginIndex: 6).getBytes(StandardCharsets.UTF_8);
    byte[] decoded = decode(base64Token);
    String token = new String(decoded, getCredentialsCharset(request));
    int delim = token.indexOf(":");
```

Reads Authorization Header

Decode the encoded username and password

After decoding, we can see What username and password Client passed

One question comes to mind is, why we need to send username and password in **Authorization header**, what not in Request body or any other way?

Possible reasons:

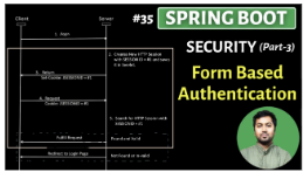
1. Standard:
As per HTTP Standardization (RFC 7617), this format is accepted, in order to make it universally accepted standard across APIs and Clients.
Otherwise, if there is no Standard follows and some send in Headers, some in body etc, then its difficult when need to deal with multiple APIs and Clients.

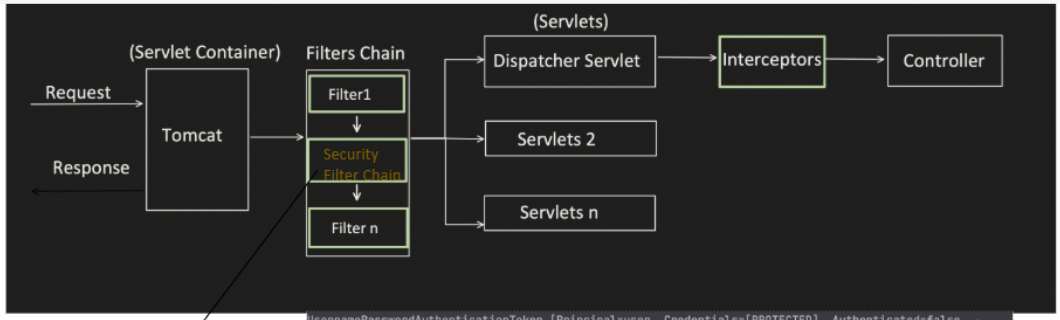
2. Security:
Web servers sometimes log the request body for debugging or analytics purpose.
But Headers are typically not logged. So this reduce the risk of exposures of client username and password.

3. Support for all HTTP request:
Apart from POST & PUT, there are HTTP requests like GET which do not accept any Request body. So with Headers for such APIs too, credentials can be sent consistently.

Now, lets understand the flow of Basic Authentication:

- **If Form Based Authentication flow is understood properly, then this flow is very similar to that.**






```
UsernamePasswordAuthenticationToken [Principal=user, Credentials=[PROTECTED], Authenticated=false, ,
,Details=null, Granted Authorities=[]]
```

This is how Authentication object looks like, currently Authenticated is false

Security Filter Chain

BasicAuthenticationFilter

1. Decode the Authorization Header

9. Save the Authentication Object in SecurityContext

SecurityContextHolder

SecurityContext

Authentication object is stored in this Context

AuthorizationFilter

2. Create and Pass the "*Authentication*" Object to

AuthenticationManager

8. Return back fully "*Authentication*" object

3. Delegates Authentication to

AuthenticationProvider

7. Updated "*Authentication*" Object

Default Implementation

ProviderManager

DaoAuthenticationProvider
(handles username/Password)

4. Incoming raw password is hashed

PasswordEncoder

5. Fetch UserDetails Like username, password, Roles etc.

<<Interface>>
UserDetailsService

Either from

InMemoryUserDetailsManager
(manage username/password in Memory)

or

JdbcUserDetailsManager

DB

(manage username/password in DB)

6. Validate Incoming Username and hashed password with Stored username and hashed password.

```
UsernamePasswordAuthenticationToken [Principal=org.springframework.security.core.userdetails.User ,
,[Username=user, Password=[PROTECTED], Enabled=true, AccountNonExpired=true, CredentialsNonExpired=true, ,
,AccountNonLocked=true, Granted Authorities=[ROLE_ADMIN]], Credentials=[PROTECTED], Authenticated=true, ,
,Details=WebAuthenticationDetails [RemoteIpAddress=0:0:0:0:0:0:0:1, SessionId=null], Granted ,
,Authorities=[ROLE_ADMIN]]
```

So, what we need to implement it?

Pom.xml

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.session</groupId>
    <artifactId>spring-session-jdbc</artifactId>
</dependency>
```

Application.properties

```
#creating username and password and assigning the ROLE to the user
spring.security.user.name=user
spring.security.user.password=pass
spring.security.user.roles=ADMIN,USER
```

```java
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception{
        http.authorizeHttpRequests(auth -> auth
                        .requestMatchers( ...patterns: "/api/users").hasAnyRole( ...roles: "USER")
                        .anyRequest().authenticated())
                .sessionManagement(session -> session
                        .sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .csrf(csrf -> csrf.disable())
                .httpBasic(Customizer.withDefaults());
        return http.build();
    }
}
```

Added for authorization

Its a stateless method

Since its stateless, CSRF is not required

Basic authentication method to be used

Disadvantages of Basic Authentication:

1. Credentials sent in every request, if HTTPS is not enforced, then it can be intercepted and then decoded.

2. If Credentials are compromised, then only way is to change the credentials.

3. Not suitable for large scale application, as sending credentials with every request is an extra overhead.
    i. As request size increases because of authorization header.
    ii. Extra work like decoding, hashing of incoming password, fetching username and password from DB, comparing etc..
    iii. DB lookup to fetch user details which increase latency too.