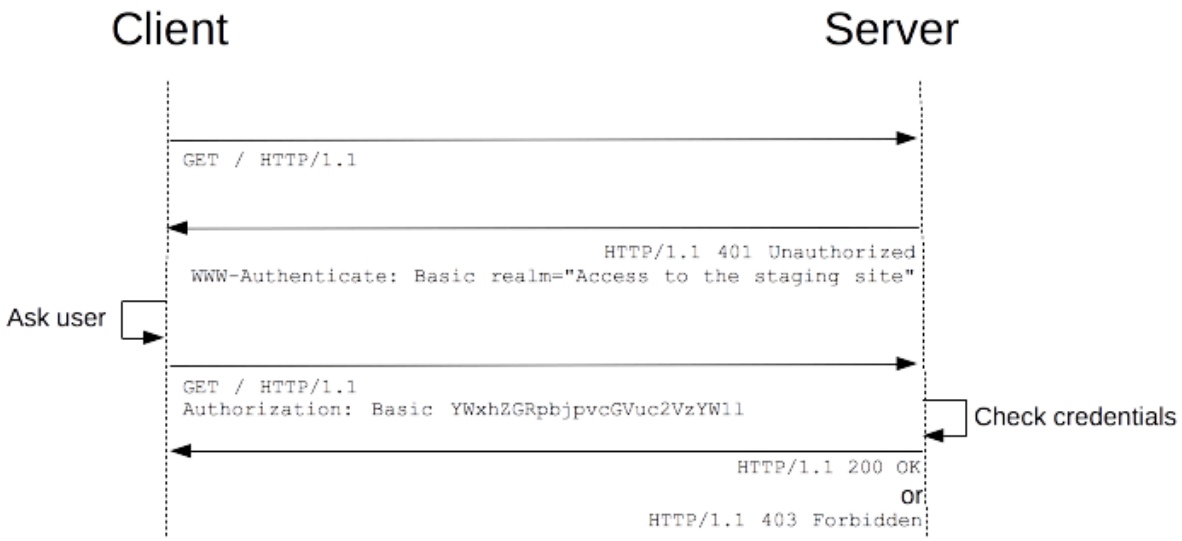
Basic Auth

WAS ?

Basis Auth ist die simpelste Methode der Authentication mit lediglich einem Username und einem Passwort.



Auf die initiale anfrage an den Server fordert dieser eine Authentication mit:

`WWW-Authenticate: Basic realm="RealmName"`

`Daraumhin sendet der Client seien Authentication in der Form`

`Authorization: Basic d2lraTpwZWRpYQ==`

Wobei der letzte Teil username:passwort, Base64 codiert darstellt. Base64 ist beim aktuellen Stand der Technik aber Klartext gleichzusetzen und der Hacker muss lediglich den Request auslesen und erhält somit die credentials.

WARUM?

Basis Authentication wird benützt um http Request zu validieren, beziehungsweise diese Anfragen auf Erlaubtheit zu prüfen.

Wo ?

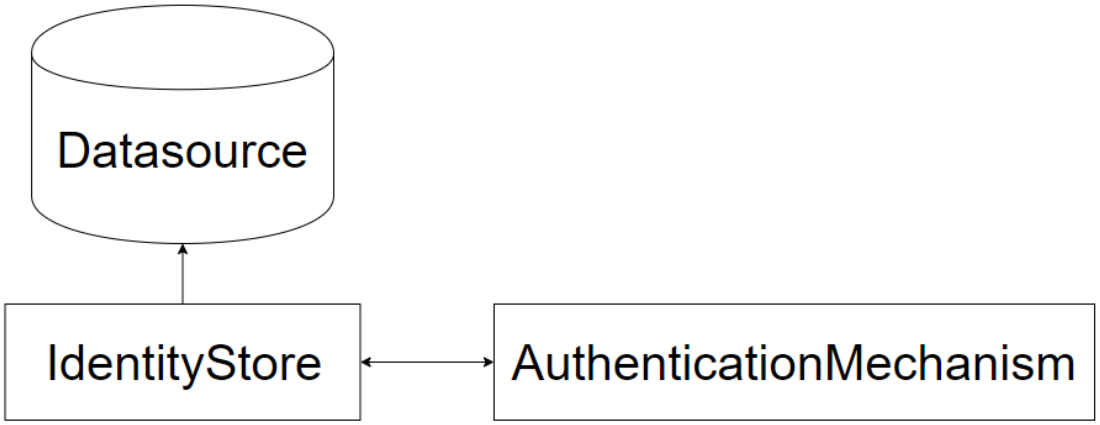
Dieses Verfahren wird heutzutage nicht mehr wirklich benutz, da die Verschlüsselung, Klartext gleichzusetzen ist. Die username:passwort kombination ist wie bereits erwähnt lediglich base64 encodier.

WIE (siehe Grafik)

Server -> `WWW-Authenticate: Basic realm="RealmName"`

`Client -> Authorization: Basic d2lraTpwZWRpYQ==`

Soteria



Aufbau

- IdentityStore der auf die Daten(z.B. Datenbank oder Liste) zugreift und somit die User Authentifiziert
- AuthenticationMechanism der den Request Authentifiziert und den Identity Store aufruft (z.B. Username and Password aus Request auslesen)

IdentityStore

```
public interface IdentityStore {
    CredentialValidationResult validate(Credential credential);
    Set<String> getCallerGroups(CredentialValidationResult result);
    int priority();
    Set<ValidationType> validationTypes();
    enum ValidationType { VALIDATE, PROVIDE_GROUPS }
}
```

Die Anfrage wird vom sogenannten IdentityStoreHandler durch die verschiedenen Identity Stores, beginnend mit dem der höchsten Priorität (von unten), geschickt. Je nach anfrage wird der erste Identity Store benutz der als ValidaitonType den nötigen Wert hat (VALIDATE oder PROVIDE_GROUPS)

Validation Types

Validate enabled 1. Methode (kann userdaten validieren)

Provide_groups enabled 1. Methode (gibt die dazugehörigen Gruppen zurück)

Rückgabewert IdentityStore

```
public class CredentialValidationResult {
    public Status getStatus() {...}
    public CallerPrincipal getCallerPrincipal() {...}
    public Set<String> getCallerGroups() {...}
    public enum Status { NOT_VALIDATED, INVALID, VALID }
}
```

Status	NOT_VALIDATED kein zuständiger IdentityStore gefunden
	INVALID abgelehnt worden
	VALID akzeptiert
getCallerPrincipal	Username
getCallerGroups	Gruppen / Rollen des validen users

Vordefinierte Identity Stores

@LdapIdentityStoreDefinition -> url + username + passwort

@DatabaseIdentityStoreDefinition -> dataSourceLookup + callerQuery = "SELECT password from USERS where name = ?"

@EmbeddedIdentityStoreDefinition

AuthenticationMechanism

```
public interface HttpAuthenticationMechanism {
    AuthenticationStatus validateRequest(
            HttpServletRequest request,
            HttpServletResponse response,
            HttpMessageContext httpMessageContext) throws AuthenticationException;
    AuthenticationStatus secureResponse(...) throws AuthenticationException;
    void cleanSubject(...) throws AuthenticationException;
}
```

Die Daten zur Validierung werden vom IdentityStore erhalten

Vorhandene Implementierungen für AuthenticationMechanism

Basic Authentication => username:password

@BasicAuthenticationMechanismDefinition

FormAuthentication => HTML-Form

@FormAuthenticationMechanismDefinition

(loginToContinue = @LoginToContinue())

@CustomFormAuthenticationMechanismDefinition

(loginToContinue=@LoginToContinue())

Rückgabewert AuthenticationMechanism

```
public interface HttpMessageContext {
    boolean isProtected();
    AuthenticationStatus responseUnauthorized();
    AuthenticationStatus responseNotFound();
    AuthenticationStatus notifyContainerAboutLogin(String callername, Set<String> groups);
    AuthenticationStatus doNothing();
}
```

keine Authentifizierung notwendig	doNothing
nicht zugriffsberechtigt	responseUnauthorized
responseNotFound	kein Identity Store
hat funktioniert	notifyContainerAboutLogin

Json WebToken

https://jwt.io/

HEADER.PAYLOAD.SIGNATURE

HEADER -> algorithmus zum encoden des secrets

PAYLOAD -> diverse Daten(CLAIMS:Ansprüche) ist lesbar -> keine geheimen Dinge

SIGNATUR -> HEADER + "." + "PAYLOAD jeweil base64 verschlüsselt -> algorithmus verschlüsseln mit secret(privateKey) -> wenn user etwas ändert und zurück schickt merkt der server das da die encodete version sich verändert hat

Predefined Klaims

iss	Issuer
exp	Expiration time
Sub	Subject
Aud	Audience
Nbf	Not Before
Iat	Issued At

Pro

+ braucht nur das secret um zu validieren(keine gespeicherten Sessions)
+ mehrer Server -> nicht mehrmals anmelden
+ Änderungen machen das Token ungültig

CON

- http://cryto.net/~joepie91/blog/2016/06/13/stop-using-jwt-for-sessions/
- groß
- können nicht gelöscht werden
- kann alte Daten enthalten

GENERELL

Authorization -> Wer bin ich ?

Authentication -> Was darf ich tun?