



UNIVERSITÀ DEGLI STUDI DI MILANO  
FACOLTÀ DI SCIENZE E TECNOLOGIE

*Sicurezza Informatica*

*Progetto in Sicurezza delle Architetture  
Orientate ai Servizi*

# **OAUTH E TWITTER**

SARA LONGO

Matricola 933528

BIAGIO DIPALMA

Matricola 938473



# Indice dei contenuti

<i>PROGETTO IN SICUREZZA DELLE ARCHITETTURE ORIENTATE AI SERVIZI</i> .....	I
<b>OAUTH E TWITTER</b> .....	I
<b>INDICE DEI CONTENUTI</b> .....	3
<b>INDICE DELLE FIGURE</b> .....	5
<b>INTRODUZIONE: OAUTH</b> .....	6
<b>CAPITOLO 1</b> .....	10
1.1 FASE 1: REGISTRAZIONE, AUTENTICAZIONE E AUTORIZZAZIONE .....	10
1.1.1 <i>Credenziali temporanee</i> .....	11
1.1.2 <i>Autorizzazione dell'utente</i> .....	12
1.1.3 <i>Rilascio delle credenziali</i> .....	13
1.2 FASE 2: RICHIESTE AUTENTICATE .....	14
1.2.1 <i>La costruzione della richiesta</i> .....	14
1.2.2 <i>La firma della richiesta</i> .....	16
<b>CAPITOLO 2</b> .....	17
2.1 CONFIG.PHP E CONNECT.PHP .....	18
2.2 REDIRECT.PHP .....	19
2.3 CALLBACK.PHP .....	21
2.4 INDEX.PHP .....	22
2.5 CLASS_ENCRYPTION-PHP E CLEARSESSIONS.PHP .....	24
<b>CAPITOLO 3</b> .....	25
3.1 BRANCH MASTER .....	25
3.1.1 <i>AESCrypto.java</i> .....	25
3.1.2 <i>WebReader.java</i> .....	26
3.1.3 <i>TwitterUtilities.java</i> .....	27

---

3.1.4 <i>Main.java</i> .....	29
3.2 BRANCH OAUTHGENERATION .....	31
<b>RIFERIMENTI SITOGRAFICI .....</b>	<b>33</b>

## Indice delle figure

Figura 0-1 – Gli attori di OAuth.	7
Figura 0-2 – Il flusso dell'autorizzazione tra il Consumer e il Service Provider.	9
Figura 2-1: Twitter Developer Application.	18
Figura 2-2: Sign in with Twitter.	19
Figura 2-3: La creazione e il salvataggio delle credenziali temporanee.	19
Figura 2-4: Autorizzazione.	20
Figura 2-5: ProgettoSOASecurity Authorization.	21
Figura 2-6: Creazione di un oggetto OAuth, richiesta dell'Access Token.	21
Figura 2-7 : Salvataggio dell'access Token e redirect alla idex.php.	22
Figura 2-8: index.php – recupero dell'Access Token e creazione di un oggetto OAuth.	22
Figura 2-9: Memorizzazione dei token criptati in un file di testo.	23
Figura 2-10: Funzione di encryption che prende in input la chiave, l'initialization value e la stringa da criptare.	24
Figura 3-1: Funzione di decryption che prende in input la chiave e la stringa da decriptare.	26
Figura 3-2: Recupero del file di testo da una pagina web.	26
Figura 3-3: WebReader.java	27
Figura 3-4: Metodo per la creazione di un tweet	28
Figura 3-5: Metodo per l'ottenimento della timeline	28
Figura 3-6: Metodo per ottenere le informazioni dell'utente.	29
Figura 3-7: metodo fetchTokens().	30
Figura 3-8: metodo createProperties().	30

---

# INTRODUZIONE: OAUTH

*Open Authorization*, più comunemente chiamato OAuth, è un protocollo di comunicazione di rete open e standard mediante il quale un'applicazione o un servizio web può gestire in modo sicuro l'accesso autorizzato ai dati sensibili.

Con il crescente successo di siti web, sono nate intorno ad essi applicazioni, sviluppate da terze parti, per utilizzare questi servizi. Parallelamente è nata la necessità di autorizzare queste applicazioni ad accedere, in modo sicuro, ai dati privati dell'utente.

L'obiettivo principale di OAuth è di consentire ad un Client l'accesso ad una risorsa, dopo essere stato approvato dal proprietario della stessa, mediante l'emissione di un *token* da parte di un Server autoritativo. Questo meccanismo è utilizzato da molte compagnie come ad esempio Google, Facebook, Amazon, Twitter, per consentire all'utente di condividere informazioni circa il proprio account con altre applicazioni o siti web e garantire a questi ultimi di accedervi senza fornire alcuna password e senza dividerne la propria identità.

Gli attori principali sono:

- L'Utente o *Resource Owner*, che garantisce l'accesso alle risorse (porzioni del proprio account), di cui è proprietario, protette sul Resource Service. È l'utente dell'applicazione iscritto sulla piattaforma del Service Provider.
- Il Service Provider o *Resource Server*: il Server utilizzato per poter accedere alle risorse o il servizio web che fornisce le informazioni.

- Il Client o *Consumer*: è l'applicazione che cerca di ottenere l'accesso all'account, tramite il permesso dell'Utente.
- L'*Authorization Server*: è il Server che mostra l'interfaccia all'interno della quale l'Utente approva o nega la richiesta di accesso. In piccoli ambienti può coincidere con il Resource Server.

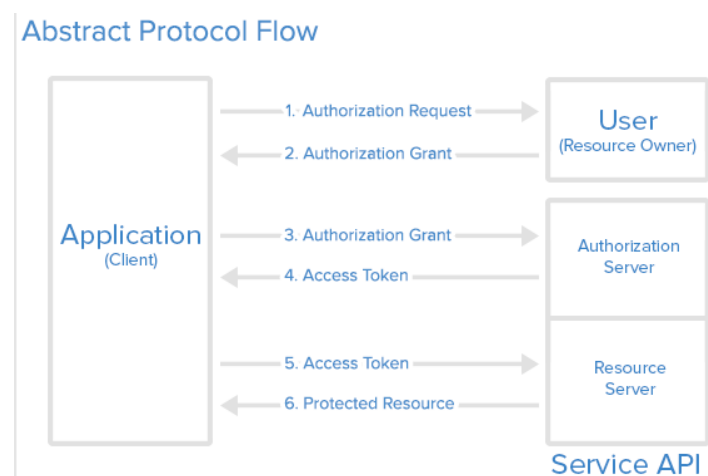


Figura 0-1 – Gli attori di OAuth.

I principali passi della comunicazione Consumer – Service Provider sono i seguenti:

1. Il Consumer registra la propria applicazione per poter usufruire del servizio predisposto dal Service Provider. Quest'ultimo gli fornisce le chiavi segrete (Consumer Key e Consumer Secret), utilizzate per lo scambio dei dati.
2. Il Consumer ridirige l'utente ad un URL predisposto dal Service Provider per la fase di autenticazione. Nella richiesta HTTP sono presenti le due chiavi ricevute in fase di registrazione e un'URL al quale l'utente deve essere ridiretto al termine dell'operazione.
3. L'utente accede alla pagina di autenticazione predisposta dal Service Provider dove inserisce le proprie credenziali di accesso, di solito username e

---

password. Di solito la pagina di autenticazione lavora con il protocollo HTTPS il quale garantisce che i dati scambiati tra browser e web server sono codificati e non vengono trasferiti in chiaro come invece avviene nel protocollo HTTP. In questo caso la comunicazione risulta molto più sicura in caso di intercettazione.

4. Il Service Provider verifica l'identità dell'utente. Se la verifica ha esito positivo, gli presenta il tipo di richiesta avanzata dal Consumer che l'utente deve approvare. Tale richiesta riguarda, di solito, la durata dell'accesso e la tipologia di informazioni che verranno rese disponibili al Consumer.
5. Il Service Provider ridirige l'utente all'URL inviata precedentemente dal Consumer. Nella richiesta HTTP è presente una stringa (detta *token*) che deve essere utilizzata successivamente dal Consumer per richiedere eventuali informazioni al Service Provider. Questo meccanismo prende il nome di *access delegation*.
6. L'utente accede nuovamente alla pagina del Consumer. Per l'utente meno esperto, il passaggio da un Server ad un altro, è praticamente trasparente.
7. Il Consumer invia una HTTP al Service Provider nella quale richiede, ad esempio, i dati anagrafici dell'utente corrente. Nella richiesta deve anche fornire il token precedentemente ricevuto. Le informazioni disponibili dipendono dal Service Provider.

Per quanto riguarda quest'ultimo punto, Twitter mostra la timeline dell'utente, Facebook espone i suoi dati anagrafici oppure i suoi ultimi messaggi in bacheca; LinkedIn espone i collegamenti di un utente oppure le sue iscrizioni ai gruppi; Google fornisce moltissime informazioni come ad esempio l'elenco dei contatti in rubrica.



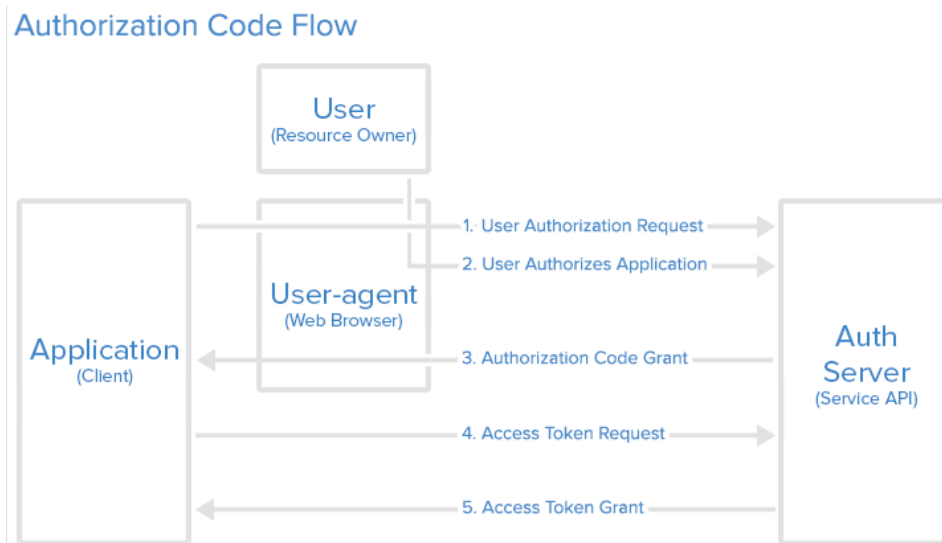


Figura 0-2 – Il flusso dell'autorizzazione tra il Consumer e il Service Provider.

Altra terminologia:

- *Risorse Protette*: insieme di dati controllati dal Service Provider, a cui il *Consumer* può accedere tramite autenticazione.
- *Consumer Key*: valore usato dal Consumer per identificarsi con il Service Provider.
- *Consumer Secret*: valore segreto usato dal Consumer per stabilire la proprietà della Consumer Key.
- *Request Token*: valore usato dal Consumer per ottenere l'autorizzazione da parte dell'utente e scambiato con un Access Token.
- *Access Token*: valore usato dal Consumer per accedere alle risorse protette, invece di usare le credenziali dell'utente sul Service Provider.

---

# CAPITOLO 1

## IL PROTOCOLLO

Il protocollo di autenticazione OAuth si divide in due fasi.

La prima parte definisce un processo, basato sul redirect da parte dell'applicazione, per permettere agli utenti finali di autorizzare il Client ad accedere alle loro risorse, attraverso l'autenticazione diretta con il Server.

La seconda parte definisce un metodo per eseguire richieste HTTP autenticate usando due set di credenziali, uno che identifica il Client che fa la richiesta e il secondo che identifica il possessore delle risorse, in nome del qual vengono effettuate le richieste.

### 1.1 Fase 1: Registrazione, autenticazione e autorizzazione

Il Consumer si iscrive come sviluppatore al Service Provider. Il Service Provider gli fornisce una coppia di chiavi: *Consumer Key* e Consumer Secret (*Shared Secret*).

L'Utente si autentica al Server attraverso l'inserimento di email e password, poi autorizza il Client attraverso l'uso dei *token* emessi dal Server su richiesta dell'Utente stesso. Per poter fornire i *token*, il Server utilizza la redirect HTTP e l>User-Agent del Resource Owner.

Per far sì che il Client possa compiere questi passi, il Server rende pubbliche le URI dei tre seguenti endpoints:

- *Temporary Credential Request*: usato dal Client per ottenere un set di credenziali temporanee.
- *Resource Owner Authorization*: endpoint al quale l'Utente è reindirizzato per concedere l'autorizzazione.
- *Token Request*: usato dal Client per scambiare il set di credenziali temporanee con il credential token.

L'autorizzazione si struttura in 3 fasi: credenziali temporanee, autorizzazione dell'utente e rilascio delle credenziali.

### **1.1.1 Credenziali temporanee**

Il Client ottiene un set di credenziali temporanee dal Server sotto forma di un identificatore ID e di un shared-secret. Le credenziali temporanee sono usate per identificare univocamente la richiesta di accesso durante tutto il processo di autenticazione.

Il Client, per poter ottenerle, deve effettuare una richiesta HTTP POST all'endpoint "Temporary credential Request". All'URI è necessario aggiungere il parametro "*oauth\_callback*", cioè un URI assoluto al quale il Server reindirizzerà l'utente quando avrà completato la fase di autorizzazione.

Per fare la richiesta, il Client si autentica usando solamente le proprie credenziali.

Il messaggio di HTTP Request sarà costituito da:

- Request Line (GET/POST, URI)
- Header (Host, User Agent, Authorization)
- Body

Il messaggio di HTTP Response sarà costituito da:

- Status Line
- Header

- 
- Body

Le credenziali temporanee vengono revocate una volta che il Client ha ottenuto il *token* per l'accesso. Per garantire maggiore sicurezza, le credenziali temporanee hanno un tempo di vita limitato. Il Server dovrebbe inoltre consentire all'Utente di revocare i *token* di accesso dopo che questi sono stati concessi al Client.

A questo punto il Server deve verificare la validità della richiesta e, in caso affermativo, rispondere al Client con un set di credenziali temporanee (sotto forma di identificatore ID e shared secret). Le credenziali temporanee sono incluse nel corpo della risposta HTTP che ritornerà con codice di status 200 (OK).

La risposta contiene i seguenti parametri richiesti:

- *oauth\_token*: identificatore delle credenziali temporanee.
- *oauth\_token\_secret*: shared secret delle credenziali temporanee.
- *oauth\_callback\_confirmed*: impostato a *True*. (Parametro usato per differenziarsi dalle versioni precedenti del protocollo).

### 1.1.2 Autorizzazione dell'utente

Il Client indirizza l'utente verso il Resource Owner Endpoint attraverso lo User Agent, dopo aver ricevuto le credenziali temporanee. Costruisce una URI attraverso una richiesta HTTP GET, aggiungendo il parametro obbligatorio "*oauth\_token*" all'endpoint del Resource Owner Endpoint: questo parametro è l'identificatore delle credenziali temporanee ottenuto nel parametro "*oauth\_token*".

Quando arriva una richiesta di autorizzazione di accesso all'Utente, il Server deve:

- presentare le informazioni sul Client che richiede l'accesso, usando l'associazione tra le credenziali temporali e l'identità del Client;
- mostrare le informazioni sul tipo di dati al quale il Client vuole accedere.

Per assicurarsi che l'Utente, che ha concesso l'autorizzazione, sia lo stesso che ritorna al Client, il Server deve generare un codice di *verifica*: passa un valore difficile da indovinare al Client tramite l'Utente. Il Server costruisce la risposta aggiungendo due parametri alla callback URI: *oauth\_token*, l'identificatore delle credenziali temporanee ricevute dal client e *oauth\_verifier*, codice di verifica.

Se il Client non fornisce una URI di *callback*, il Server dovrebbe visualizzare il valore del codice di verifica e dare indicazioni all'Utente su come informare manualmente il Client che l'autorizzazione è completata.

### **1.1.3 Rilascio delle credenziali**

Il Client richiede un set di token al Server che gli permetteranno di accedere alle risorse protette dell'Utente. Li ottiene mediante una richiesta HTTP POST autenticata all'endpoint per le *Token Requests*. Costruisce una richiesta URI aggiungendo il parametro *oauth\_verifier*, cioè il codice di verifica ricevuto precedentemente.

Al momento della richiesta, il Client viene autenticato utilizzando sia le sue credenziali sia quelle temporanee che sono trasmesse mediante il parametro "*oauth\_token*".

Visto che le credenziali vengono trasmesse come semplice testo, il Server deve richiedere l'uso di un meccanismo a livello di trasporto come *TLS* o *SSL4* per garantire la sicurezza dei dati trasmessi.

Il Server deve:

- verificare la validità della richiesta, assicurandosi che l'Utente abbia autorizzato il rilascio dei token;
- garantire che le credenziali temporanee non siano mai state usate prima e non siano scadute;

- 
- verificare il codice di verifica inviato dal Client.

Se la richiesta è valida e autorizzata, le credenziali token saranno incluse nel corpo della risposta HTTP. La risposta del Server contiene il parametro *oauth\_token*, cioè l'identificativo del token e *oauth\_token\_secret*, cioè lo shared-secret del token.

Il Server deve mantenere lo scopo, la durata, e altri attributi del token approvati dal Resource Owner e applicare queste restrizioni quando riceve una richiesta dal Client fatta con le credenziali token concesse.

## 1.2 Fase 2: richieste autenticate

A questo punto il Client può eseguire richieste autenticate in nome del Resource Owner; ad ogni richiesta deve presentare due set di credenziali: uno che lo identifichi e l'altro che identifichi l'Utente.

Le credenziali del Client sono costituite da coppia di chiavi RSA o una coppia identificatore/shared-secret.

### 1.2.1 La costruzione della richiesta

La richiesta è costituita dai seguenti parametri:

- *oauth\_consumer\_key*: l'identificatore delle credenziali Client.
- *oauth\_token*: Il valore del token usato per associare la richiesta col Resource Owner.
- *oauth\_signature\_method*: il nome del metodo con cui vengono firmate le richieste.

- *oauth\_timestamp*: Il valore del timestamp (intero positivo espresso in secondi). Il parametro può essere omissso se si usa il metodo di firma "PLAINTEXT".
- *oauth\_nonce*: valore del “nonce”, una stringa casuale, generata univocamente dal Client per permettere al Server di verificare che una richiesta non sia mai stata fatta prima e aiutare a prevenire i *Replay Attacks* su canali non sicuri. Il valore della nonce deve essere unico tra tutte le richieste con lo stesso timestamp, credenziali client e condizioni token. Per evitare di dover generare un numero infinito di nonce per controlli futuri, i server possono scegliere di restringere il lasso di tempo dopo il quale un vecchio timestamp è rifiutato. Il parametro può essere omissso se si usa il metodo di firma "PLAINTEXT".
- *oauth\_version*: parametro opzionale. Se presente deve essere settato a 1.0. Fornisce la versione del protocollo di autenticazione.

I passaggi che il Client deve eseguire per effettuare una richiesta autenticata sono quattro:

1. Il Client assegna i valori ad ognuno dei parametri;
2. I parametri sono aggiunti alla richiesta. Ogni parametro non deve comparire più di una volta per richiesta. Viene costruita una stringa che sarà l'input degli algoritmi di firma: HMAC-SHA1 per la chiave simmetrica, RSA-SHA1 per la chiave pubblica/privata e PLAIN-TEXT senza firma.
3. Il Client calcola e assegna il valore del parametro "*oauth\_signature*" e lo aggiunge alla richiesta, usando il *Client Secret* e il *Token Secret*. L'output di questi algoritmi sarà il valore del parametro "*oauth\_signature*".
4. Il Client manda la richiesta HTTP autenticata al Server.

---

### 1.2.2 La firma della richiesta

Una volta che il Server ha ricevuto la richiesta, deve validarla, cioè deve ricalcolare la firma indipendentemente e confrontarla con il parametro `"oauth_signature"` inviato dal Client.

Inoltre, se è stato usato `"HMAC-SHA1"` o `"RSA-SHA1"` come metodo di cifratura, deve assicurarsi che la combinazione `nonce/timestamp/token` non sia mai stata usata. Per ultimo deve verificare lo scopo del token e se quest'ultimo è ancora valido.

Se la richiesta fallisce, il Server dovrebbe rispondere con l'appropriato codice di stato della richiesta HTTP.

Se la richiesta è valida, il Server concede al Client le risorse. Il Client deve provare che sia il legittimo proprietario delle credenziali *Consumer Key e Token*, che invia al Server per accedere alle risorse protette dell'Utente. Per far ciò usa lo `shared-secret` o la chiave RSA che sono parte di ogni set di credenziali. Queste sono usate in abbinamento a metodi di firma come `"HMAC-SHA1"`, `"RSA-SHA1"` e `"PLAINTEXT"`.

Come input per i metodi di firma, viene generata una stringa che è una concatenazione di diversi elementi della richiesta HTTP. Questa stringa conterrà i seguenti parametri:

- il tipo della richiesta HTTP (`"GET"`, `"POST"`, etc.).
- il campo `"Host"` dell'header della richiesta HTTP.
- il path e le componenti query della richiesta URI.
- i parametri di protocollo escluso `"oauth_signature"`. Il metodo di cifratura basato su stringa non copre l'intera richiesta HTTP come per esempio il corpo della richiesta. Per questo motivo il Server non può verificare l'autenticità dei parametri esclusi a meno di usare un canale sicuro per la trasmissione.



## CAPITOLO 2


### GLI SCRIPT IN PHP

Al fine di registrare un'applicazione per Developers su Twitter, è stato creato un account, per poter ricorrere all'uso di OAuth, che introduce numerosi vantaggi in termini di sicurezza, come ad esempio il fatto che si possano modificare password senza che ciò abbia impatto sull'applicazione o che le credenziali di accesso al social network non viaggino più con le richieste http/https.

È stato utilizzato *Altervista* come provider per l'hosting web, in modo da impostare la "Website URL" e la "Callback URL", fondamentali per il funzionamento di OAuth.

Sono stati poi recuperati i due token (ACCESS TOKEN E ACCESS TOKEN SECRET) e le due chiavi (CONSUMER KEY E CONSUMER SECRET).

---

**App icon**  
App icon is default, click edit to upload.

**App Name**  
ProgettoSOASecurity

**Description**  
It's a Cyber Security University project for SOA Security course.

**Website URL**  
<https://progettosoasecurity.altervista.org>

**Sign in with Twitter**  
Enabled

**Callback URL**  
<https://progettosoasecurity.altervista.org/callback.php>

Figura 2-1: Twitter Developer Application.

Dapprima è stata creata una directory contenente il codice PHP, costituita da: *config.php*, *connect.php*, *callback.php*, *redirect.php*, *index.php*, *clearsessions.php* e *class\_encryption*.

## 2.1 Config.php e connect.php

Nel file *config.php* sono state definite le due chiavi *CONSUMER\_KEY* e *CONSUMER\_SECRET*, ottenute attraverso il Developer Account, e la *CALLBACK* all'indirizzo <https://progettosoasecurity.altervista.org/callback.php>.

La pagina *connect.php* è la prima ad essere visualizzata, dato che mostra il bottone della login attraverso Twitter.

All'interno del file si controlla che i due parametri (le due chiavi) siano stati definiti in maniera corretta, portando ad una situazione di errore in caso contrario.

L'utente viene portato a fare una *token request*, attraverso il click del bottone, iniziando il processo di redirect.

## Welcome to a Twitter OAuth PHP example



Figura 2-2: Sign in with Twitter.

### 2.2 Redirect.php

All'interno della *redirect.php* viene creato un oggetto OAuth con le credenziali del Client; poi, vengono create e salvate le credenziali temporanee:

```
//CREAZIONE OGGETTO OAUTH CON CREDENZIALI CLIENT
$connection = new TwitterOAuth(CONSUMER_KEY, CONSUMER_SECRET);

//CREDENZIALI TEMPORANEE
$request_token = $connection->getRequestToken(OAUTH_CALLBACK);

//SALVATAGGIO DELLE CREDENZIALI
$_SESSION['oauth_token'] = $token = $request_token['oauth_token'];
$_SESSION['oauth_token_secret'] = $request_token['oauth_token_secret'];
```

Figura 2-3: La creazione e il salvataggio delle credenziali temporanee.

Il Client, per poter ottenerle, deve effettuare una richiesta HTTP POST all'endpoint "Temporary credential Request". All'URI è necessario aggiungere il

---

parametro “*oauth\_callback*”, cioè un URI assoluto al quale il Server redirezionerà l'utente quando avrà completato lo step di autorizzazione.

GetRequestToken è una funzione che prende in input il parametro OAUTH\_CALLBACK e lo utilizza nella richiesta GET per poter creare un OAuth Consumer con i parametri *oauth\_token* e *oauth\_token\_secret*.

Le credenziali temporanee vengono revocate una volta che il Client ha ottenuto il *l'access token*. Per garantire maggiore sicurezza, le credenziali temporanee hanno un tempo di vita limitato.

```
switch ($connection->http_code) {
    case 200:
        //URL AUTORIZZATA
        //getAuthorizeURL è in twitteroauth.php
        $url = $connection->getAuthorizeURL($token);
        header('Location: ' . $url);
        break;
    default:
        //NOTIFICA SE QUALCOSA VA STORTO
        echo 'Could not connect to Twitter. Refresh the page or try again later.';
}
```

Figura 2-4: Autorizzazione.

La funzione *getAuthorizeURL* prende in input il token e il flag "sign in with twitter" impostato a true.

A questo punto il Server deve verificare la validità della richiesta e, in caso affermativo, rispondere al Client con un set di credenziali temporanee (sotto forma di identificatore ID e shared secret).

Questa pagina è fondamentale perché mostra *l'Authorization Link* che dovrà essere approvato dall'utente se la URL è autorizzata:

## Authorize ProgettoSOASecurity to use your account?



### This application will be able to:

- Read Tweets from your timeline.
- See who you follow, and follow new people.
- Update your profile.
- Post Tweets for you.

### Will not be able to:

- Access your direct messages.
- See your email address.
- See your Twitter password.



ProgettoSOASecurity

progettosoasecurity.altervista.org

It's a Cyber Security University project for SOA Security course.

Figura 2-5: ProgettoSOASecurity Authorization.

## 2.3 Callback.php

La *callback.php* permette di creare un oggetto OAuth con il *CONSUMER\_KEY*, il *CONSUMER\_SECRET*, l' *APP\_TOKEN* e l' *APP\_TOKEN\_SECRET* e in seguito di fare una richiesta per recuperare l'*ACCESS\_TOKEN*, che viene successivamente salvato:

```
//SE IL TOKEN è VECCHIO, FARE IL REDIRECT ALLA PAGINA
if (isset($_REQUEST['oauth_token']) && $_SESSION['oauth_token'] !== $_REQUEST['oauth_token']) {
    $_SESSION['oauth_status'] = 'oldtoken';
    header('Location: ./clearsessions.php');
}

//CREAZIONE OGGETTO OAUTH CON APP KEY/SECRET E TOKEN KEY/SECRET
$connection = new TwitterOAuth(CONSUMER_KEY, CONSUMER_SECRET, $_SESSION['oauth_token'], $_SESSION['oauth_token_secret']);

//RICHIESTA DI ACCESS TOKEN
//twitteroauth.php
$access_token = $connection->getAccessToken($_REQUEST['oauth_verifier']);
```

Figura 2-6: Creazione di un oggetto OAuth, richiesta dell'Access Token.

---

La funzione `getAccessToken` prende in input il parametro `oauth_verifier` settato a `false`, poi fa una `OAuthRequest` di tipo `get` usando come parametro l'`oauth_verifier` stesso.

Crea poi un `OAuthConsumer` avente come token l'`oauth_token` o l'`oauth_token_secret`.

```
//SALVATAGGIO DELL'ACCESS TOKEN
$_SESSION['access_token'] = $access_token;

unset($_SESSION['oauth_token']);
unset($_SESSION['oauth_token_secret']);
|
if (200 == $connection->http_code) {

    //L'UTENTE È STATO VERIFICATO E L'ACCESS TOKEN PUÒ ESSERE SALVATO PER USO FUTURO
    $_SESSION['status'] = 'verified';
    header('Location: ./index.php');
} else {
    header('Location: ./clearsessions.php');
}
```

Figura 2-7 : Salvataggio dell'access Token e redirect alla `index.php`.

A questo punto l'utente è stato verificato e viene reindirizzato alla pagina `index.php`.

## 2.4 Index.php

All'interno del file `index.php` si crea un oggetto `OAuth Twitter` con i quattro parametri ottenuti:

```
//SE GLI ACCESS TOKEN NON SONO DISPONIBILI, FARE LA REDIRECT ALLA CONNECT PAGE
if (empty($_SESSION['access_token']) || empty($_SESSION['access_token']['oauth_token']) || empty($_SESSION['access_token']['oauth_token_secret'])) {
    header('Location: ./clearsessions.php');
}

//GET USER ACCESS TOKENS
$access_token = $_SESSION['access_token'];

//CREAZIONE OGGETTO TWITTER OAUTH CON I TOKEN
$connection = new TwitterOAuth(CONSUMER_KEY, CONSUMER_SECRET, $access_token['oauth_token'], $access_token['oauth_token_secret']);
```

Figura 2-8: `index.php` – recupero dell'Access Token e creazione di un oggetto `OAuth`.

Dopo di che, si applica una cifratura dei token grazie al cifrario simmetrico AES in modalità CBC (*Cipher Block Chaining*), attraverso l'uso di una chiave e di un Initialization Vector e si memorizzano all'interno di un file di testo, che viene ripulito ogni volta che si effettua una nuova login.

```
//ISTANZIO LA CLASSE PER LA CIFRATURA TOKEN
$key_enc=new PHP_AES_Cipher();
$secre_enc=new PHP_AES_Cipher();
$tok_enc=new PHP_AES_Cipher();
$tok_sec_enc=new PHP_AES_Cipher();

$chiave = "dipalmalongoprogram";
$iv = "progettosoasecur";

//MEMORIZZAZIONE DEI TOKEN IN UN FILE
$file =fopen("token.txt", "w+");
fwrite($file, $key_enc->encrypt($chiave, $iv, CONSUMER_KEY).",".");
fwrite($file, $secre_enc->encrypt($chiave, $iv, CONSUMER_SECRET).",".");
fwrite($file, $tok_enc->encrypt($chiave, $iv, $access_token['oauth_token']).",".");
fwrite($file, $tok_sec_enc->encrypt($chiave, $iv, $access_token['oauth_token_secret']));
fclose($file);
```

Figura 2-9: Memorizzazione dei token criptati in un file di testo.

---

## 2.5 Class `_encryption.php` e `clearsessions.php`

`Class_encryption.php` contiene la classe in PHP utilizzata per creare una funzione di encryption e decryption con AES a 128 bit di chiave (16 bytes), in base 64.

```
class PHP_AES_Cipher {

    private static $OPENSSL_CIPHER_NAME = "aes-128-cbc"; //nome del cifrato
    private static $CIPHER_KEY_LEN = 16; //128 bit

    static function encrypt($key, $iv, $data) {
        if (strlen($key) < PHP_AES_Cipher::$CIPHER_KEY_LEN) {
            $key = str_pad($key, PHP_AES_Cipher::$CIPHER_KEY_LEN, "0"); //0 padding
        } else if (strlen($key) > PHP_AES_Cipher::$CIPHER_KEY_LEN) {
            $key = substr($key, 0, PHP_AES_Cipher::$CIPHER_KEY_LEN); //troncato a 16 bytes
        }

        $encodedEncryptedData = base64_encode(openssl_encrypt($data, PHP_AES_Cipher::$OPENSSL_CIPHER_NAME, $key, OPENSSL_RAW_DATA, $iv));
        $encodedIV = base64_encode($iv);
        $encryptedPayload = $encodedEncryptedData." ".$encodedIV;

        return $encryptedPayload;
    }
}
```

Figura 2-10: Funzione di encryption che prende in input la chiave, l'initialization value e la stringa da criptare.

Il file `clearsessions.php` è stato inserito per poter caricare e pulire la sessione e per poter ritornare alla `connect.php`.



# CAPITOLO 3

## GLI SCRIPT IN JAVA

### 3.1 Branch master

Per quanto concerne lo script in Java, esso funge da Consumer Server: l'obiettivo di questo script è quello di recuperare i token (scritti in un file di testo all'URL [www.progettosasec.altervista.org/token.txt](http://www.progettosasec.altervista.org/token.txt) in forma cifrata) e, attraverso gli stessi, far eseguire delle azioni da parte dell'utente, che in questo contesto simula il Consumer Server.

Lo script si compone principalmente delle classi: *AESCrypto.java*, *WebReader.java*, *TwitterUtilities.java* e *Main.java*.

#### 3.1.1 AESCrypto.java

Questa classe racchiude le funzioni di cifratura. In particolare, il metodo di interesse è “decrypt”, il duale del metodo di cifratura scritto in PHP. Il sistema di cifratura impiegato è AES, con chiave “*dipalmalongoprogram*” e IV “*progettosasec*”, entrambi di 128 bit.

La realizzazione delle funzioni di cifratura e decifratura è stata possibile utilizzando le librerie Javax: le classi *SecretKeySpec* e *IvParameterSpec* sono state utili per trasformare le stringhe *key* e *IV* in bit.

```

/**
 * Decrypt data using AES Cipher (CBC) with 128 bit key
 *
 * @param key - key to use should be 16 bytes long (128 bits)
 * @param data - encrypted data with iv at the end separate by :
 * @return decrypted data string
 */
public static String decrypt(String key, String data) {
    try {
        String[] parts = data.split( regex: ":" );

        IvParameterSpec iv = new IvParameterSpec(Base64.getDecoder().decode(parts[1]));
        SecretKeySpec skeySpec = new SecretKeySpec(key.getBytes( charsetName: "UTF-8" ), algorithm: "AES");

        Cipher cipher = Cipher.getInstance(AESCrypto.CIPHER_NAME);
        cipher.init(Cipher.DECRYPT_MODE, skeySpec, iv);

        byte[] decodedEncryptedData = Base64.getDecoder().decode(parts[0]);

        byte[] original = cipher.doFinal(decodedEncryptedData);

        return new String(original);
    } catch (Exception ex) {
        ex.printStackTrace();
    }

    return null;
}

```

Figura 3-1: Funzione di decryption che prende in input la chiave e la stringa da decriptare.

### 3.1.2 WebReader.java

Questa classe è stata utile per la costruzione di una sorta di WebReader, ovvero di una componente che potesse leggere il contenuto del *file token.txt*, attraverso un semplice scanner.

```

/**
 * Questo metodo recupera il testo da una pagina web
 * @param url della pagina web
 * @return line - stringa del testo recuperato
 * @throws IOException
 */
public static String fetchPage(URL url) throws IOException {
    Scanner s = new Scanner(url.openStream());
    String line = s.nextLine();
    return line;
}

```

Figura 3-2: Recupero del file di testo da una pagina web.

Nel caso in cui il file *token.txt* venisse scritto dal PHP, ovvero nel caso in cui l'accesso dell'utente all'applicazione Twitter sia andata a buon fine, la stringa, contenente i quattro parametri separati da virgola, viene "splittata"; ogni token è decifrato ed inserito nell'apposito array.

```
/**
 * Questo metodo invece elabora i token e li inserisce in un array
 * @param line - stringa recuperata dalla lettura del testo nella pagina web
 * @return tokens - array dei tokens decifrati
 */
public static String[] parseText(String line) throws IOException {

    String tokens [] = line.split( regex: "," );

    //creazione del file delle property per la libreria twitter4j
    File file = new File( pathname: "twitter4j.properties");

    //Create the file
    if (file.createNewFile())
    {
        System.out.println("Il file delle credenziali è stato generato!");
    } else {
        //se il file esiste, va ripulito e riscritto
        FileWriter writer = new FileWriter(file);
        writer.write( str: "debug=true\noauth.consumerKey="+tokens[0]+"\\oauth.consumerSecret="+tokens[1]+
            "\\oauth.accessToken="+tokens[2]+"\\oauth.accessTokenSecret="+tokens[3]+"\\n");
        writer.close();
    }

    //Write Content
    FileWriter writer = new FileWriter(file);
    writer.write( str: "debug=true\noauth.consumerKey="+tokens[0]+"\\oauth.consumerSecret="+tokens[1]
        +"\\oauth.accessToken="+tokens[2]+"\\oauth.accessTokenSecret="+tokens[3]+"\\n");
    writer.close();

    return tokens;
}
/*
ORDINE DEI TOKEN RECUPERATI
1 - Consumer Key
2 - Consumer Secret
3 - Access Token
4 - Access Token Secret
*/
}
```

Figura 3-3: WebReader.java

### 3.1.3 TwitterUtilities.java

TwitterUtilities.java contiene i metodi per l'esecuzione di azioni proposte all'utente, attraverso un menù all'interno della classe Main.java. Queste operazioni possono essere eseguite dal Consumer Server solo dopo aver ottenuto i token. La libreria Twitter4J ha permesso la costruzione di queste utilities.

---

Le azioni implementate sono:

- Postare un tweet

```
* Questo metodo serve per postare un tweet
* @param tweet
* @return
* @throws TwitterException
*/
public static String createTweet(String tweet) throws TwitterException {
    ConfigurationBuilder cb = new ConfigurationBuilder();
    TwitterFactory tf = new TwitterFactory(cb.build());
    Twitter twitter = tf.getInstance();
    Status status = twitter.updateStatus(tweet);
    return status.getText();
}
```

Figura 3-4: Metodo per la creazione di un tweet

- Ottenere la timeline della home (tweet postati da altri utenti)

```
/**
 * Questo metodo stampa la timeline (bacheca) dei tweet dell'utente
 * @throws TwitterException
 */
public static void getHomeTimeline() throws TwitterException {
    Twitter twitter = TwitterFactory.getSingleton();
    List<Status> statuses = twitter.getHomeTimeline();
    System.out.println("Mostro la home timeline");
    for (Status status : statuses) {
        System.out.println(status.getUser().getName() + ":" +
            status.getText());
    }
}
```

Figura 3-5: Metodo per l'ottenimento della timeline

- Ottenere le informazioni dell'utente

```
* Questo metodo mostra le info dell'utente
* @throws TwitterException
*/
public static void infoUtente () throws TwitterException{
    Twitter twitter = new TwitterFactory().getInstance();
    User user = null;
    try {
        user = twitter.showUser(TwitterUtilities.NOME_UTENTE);
    } catch (TwitterException e) {
        e.printStackTrace();
    }
    if (user.getStatus() != null) {
        System.out.println("@ " + user.getScreenName() + " - " + user.getStatus().getText());
    } else {
        // the user is protected
        System.out.println("Nome Utente= @" + user.getScreenName());
    }
    System.exit(0);
}
```

Figura 3-6: Metodo per ottenere le informazioni dell'utente.

In questa circostanza è bene spiegare che, inizialmente, l'intento era quello di implementare, in modo del tutto autonomo, del codice che potesse creare delle richieste POST e GET a Twitter al fine di realizzare le funzioni sopracitate.

Il motivo per cui non è stato possibile raggiungere l'obiettivo è la scarsa documentazione ufficiale sulla generazione della firma. I problemi riscontrati sono discussi nella sezione riguardante il branch *oauthGeneration*.

### 3.1.4 Main.java

Questa è la classe principale che contiene il main, ovvero la parte eseguibile dello script.

Il metodo *fetchTokens()* sfrutta la classe *WebReader.java* per recuperare la stringa posta nel file token.txt e scrive nelle variabili statiche i valori dei token, dopo averli decifrati.

```

/**
 * Questo metodo recupera i token da remoto e li decifra,
 * a questo punto li salva nelle variabili statiche di questa classe.
 * @throws IOException
 */
private static void fetchTokens() throws IOException {

    //Recupero i token dal sito web e li decritto
    String textFromPage = WebReader.fetchPage(urlCredentials);
    String tokens[] = WebReader.parseText(textFromPage);

    //inserisco i token nei campi privati di questa classe e applico la decifratura
    oauth_cons_secret = AESCrypto.decrypt(AESCrypto.key, tokens[1]);
    oauth_cons_token = AESCrypto.decrypt(AESCrypto.key, tokens[0]);
    oauth_user_secret = AESCrypto.decrypt(AESCrypto.key, tokens[3]);
    oauth_user_token = AESCrypto.decrypt(AESCrypto.key, tokens[2]);
}

```

Figura 3-7: metodo fetchTokens().

Il metodo *createProperties* si occupa di creare un file (Twitter4J.properties) all'interno del quale sono contenuti dei campi, tra i quali tutti i token rilasciati da Twitter, affinché la libreria Twitter4J\* possa funzionare correttamente. Questo file ha una composizione interna strutturata in base alle linee guida della libreria stessa, affinché possa funzionare. Tale file delle proprietà è stato estromesso dal versioning di Github.

```

/**
 * Questo metodo serve per generare il file twitter4j.properties: questo file di proprietà
 * è utile alla libreria per eseguire le operazioni con twitter (GET e POST)
 * @throws IOException
 */
public static void createProperties() throws IOException {
    //creazione del file delle property per la libreria twitter4j
    File file = new File( pathname: "twitter4j.properties");

    //Create the file
    if (file.createNewFile())
    {
        System.out.println("File is created!");
    } else {

        //se il file esiste, va ripulito e riscritto
        FileWriter writer = new FileWriter(file);
        writer.write( str: "");
        writer.write( str: "debug=true\noauth.consumerKey="+oauth_cons_token+"\noauth.consumerSecret="+oauth_cons_secret+
            "\noauth.accessToken="+oauth_user_token+"\noauth.accessTokenSecret="+oauth_user_secret+"\n");
        writer.close();
        System.out.println("File delle credenziali aggiornato!");
    }

    //Write Content
    FileWriter writer = new FileWriter(file);
    writer.write( str: "debug=true\noauth.consumerKey="+oauth_cons_token+"\noauth.consumerSecret="+oauth_cons_secret+
        "\noauth.accessToken="+oauth_user_token+"\noauth.accessTokenSecret="+oauth_user_secret+"\n");
    writer.close();
}

```

Figura 3-8: metodo createProperties().

Quando l'utente avvia lo script, viene mostrato il link al quale recarsi per effettuare l'accesso con Twitter e si attende un input da tastiera da inserire solo dopo che l'utente ha completato il login. A questo punto saranno recuperate le credenziali e mostrate all'utente le 3 azioni eseguibili, implementate dalla classe *Twitterutilities.java*. Qualsiasi azione sia messa in atto, lo script mostrerà i dettagli della richiesta http inviata a Twitter, oltre che all'output relativo all'azione eseguita.

## 3.2 Branch oauthGeneration

Il branch "*oauthGeneration*" riporta il codice che avrebbe dovuto permettere l'implementazione di questa funzionalità.

L'obiettivo della classe *OauthClient.java* era quello di generare degli oggetti "oauthClients", con il compito di creare le componenti previste dal meccanismo di OAuth implementato da Twitter, contenute in una stringa.

La stringa da generare contiene:

- Consumer Key
- Consumer Token
- Access Token
- Access Token Secret
- Nonce (valore generato in maniera casuale, utile a identificare la richiesta)
- Timestamp
- Versione di OAuth
- Firma

Per quanto concerne la firma, essa è generata attraverso l'algoritmo di cifratura HMAC-SHA1 che cifra una *signatureBaseString* (stringa che contiene quasi tutte le componenti elencate in precedenza) con una *signingKey* composta dalla concatenazione del Consumer Key Secret e l'Access Token Secret.

---

La stringa generata in questo modo viene inserita nell'header denominato Authorization della richiesta POST o GET che viene inoltrata a Twitter. A questa richiesta vengono inoltre aggiunti, ma non sono obbligatori, indicazioni sull'host e i cookies.

Per l'implementazione abbiamo fatto riferimento alla documentazione disponibile sul portale developer di Twitter<sup>[3]</sup>, seguendo le indicazioni sulla struttura delle richieste e della firma, non riscontrando successo. Eseguendo delle prove con altri strumenti come Postman<sup>[4]</sup> abbiamo notato che vi erano presenti altri token (Postman ne inserisce uno proprietario); allo stesso modo anche nelle richieste della libreria Twitter4J vi erano dei token ad hoc. Perciò, non trovando altro riscontro online, abbiamo deciso di utilizzare la libreria sopracitata, indicata anche da Twitter come la migliore per la programmazione in Java.



# RIFERIMENTI SITOGRAFICI

- [1] Twitter Developer: <https://developer.twitter.com/en.html>
- [2] Twitter4j: <http://twitter4j.org/en/>
- [3] Documentazione costruzione richieste Twitter:  
<https://developer.twitter.com/en/docs/basics/authentication/guides/authorizing-a-request.html>
- [4] Postman app: <https://www.getpostman.com>

Il progetto è su <http://progettosoasecurity.altervista.org/connect.php>, mentre il codice è disponibile nella directory Github <https://github.com/dipalmabiagio/progettoSOA>.