Università degli studi di Perugia

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea in Informatica

Il protocollo OAuth

STUDENTI

Luca Mancini
Riccardo Queri

PROFESSORE

Stefano Bistarelli

Anno Accademico 2009/2010

1. Introduzione

OAuth, acronimo di Open Authorization, è un protocollo open che permette l'uso autorizzato e sicuro di API (Application Programming Interface) da parte di applicazioni su sistemi desktop, web e mobile.

Lo sviluppo è iniziato nel novembre 2006 ad opera di un gruppo di programmatori web tra cui Blaine Cook, che all'epoca stava sviluppando il sistema Twitter OpenID1. L'idea fu quella di utilizzare OpenID in combinazione con le API Twitter per delegare l'autenticazione a Twitter stesso. Da questa idea, nel luglio 2007 fu rilasciata la versione finale di OAuth Core 1.0.

Con il crescente successo di siti web come Twitter, Facebook, Google etc. sono nate intorno ad essi applicazioni, sviluppate da terze parti, per utilizzare questi servizi; basti pensare ai vari client mobile per i social network. Parallelamente è quindi nata la necessità di autorizzare queste applicazioni ad accedere, in modo sicuro, ai dati privati dell'utente.

OAuth non è un concetto nuovo ma è la standardizzazione e la combinazione di diversi protocolli di settore ben collaudati. É simile ad altri protocolli attualmente in uso come Google AuthSub, AOL OpenAuth, Yahoo BBAuth, Flickr API, Amazon Web Services API, etc. Ognuno di questi protocolli fornisce un metodo proprietario per scambiare le credenziali dell'utente con un access token o ticker. OAuth è stato creato studiando attentamente ognuno di questi protocolli con l'obbiettivo di creare qualcosa che permetta una transazione agevole tra tutti questi servizi, integrando i migliori elementi di ognuno di essi.
Un'area dove OAuth è più evoluto di alcuni degli altri protocolli e servizi è la sua gestione diretta dei servizi non-web, infatti ha un supporto incorporato per le applicazioni desktop, dispositivi mobili, e naturalmente siti web.

1.1 Client-Server vs OAuth

Client-Server
Il modello tradizionale di autenticazione Client-Server usa le credenziali dell'utente (tipicamente username e password) per accedere alle risorse private che sono ospitate nel server. Con questo metodo di autenticazione, l'uso di applicazioni sviluppate da terzi parti pone chiari problemi di sicurezza:

1. la password non è più un segreto tra servizio e utente, viene salvata anche dall'applicazione.
2. l'applicazione client a totale accesso alle informazioni riservate.
3. l'autorizzazione rimane in possesso dell'applicazione client fino al cambiamento della password da parte dell'utente.

OAuth
L'idea proposta da OAuth, per risolvere questo problema, è quella di introdurre un terzo ruolo oltre ai due classici, client e server: il proprietario delle risorse (resource owner).
Nel modello OAuth il client, che non è il proprietario delle risorse ma è colui che agisce in suo nome, deve richiede il permesso all'utente per accedere alle sue risorse. Inoltre OAuth permette al

server di verificare non solo che il client abbia l'autorizzazione, da parte del proprietario delle risorse, ma anche l'identità del client che fa la richiesta.

Quindi, affinché il client possa accedere alle risorse, in primo luogo deve ottenere il permesso dal proprietario della risorsa. Questa autorizzazione viene espressa in forma di token (gettone) ed uno shared-secret. Il cuore del protocollo è proprio il token, che permette appunto di gestire gli accessi senza la condivisione della password e diversamente dalle credenziali dell'utente, può essere rilasciato con delle restrizioni e/o con durata limitata, nonché essere revocato in maniera indipendente dalla volontà del client.

È la stessa idea delle "valet key" delle macchine di lusso: una chiave speciale per i parcheggiatori, che diversamente dalla chiave regolare, non permette alla macchina di essere guidata per più di uno o due chilometri. Alcune “valet key” non permettono per esempio di aprire il bagagliaglio, mentre altre bloccano l'accesso al telefono di bordo. Indipendentemente dalle restrizioni imposte dalla “vallet key”, l'idea è molto inteligente: fornire l'accesso limitato alla macchina con una chiave speciale e usare la chiave regolare per sblocca tutto.

Un esempio concreto: un utente web (resource owner) può concedere ad un servizio di stampa (client) l'accesso alle sue foto salvate in un servizio di foto sharing (server) come Flicker, senza condividere la sua username e password con il servizio di stampa. Perciò, si autentica direttamente con il sito di foto sharing che rilascia al servizio di stampa delle credenziali per poter accedere alle foto.

1.2 Terminologia

Prima di entrare nel dettaglio ulcuni termini utili che verranno utilizzati nel seguito:

- *Service Provider*: è il sito o servizio web, all'interno del quale le risorse sono memorizzate.
- *Resource Owner*: colui che possiede un account nel *Service Provider*. In sostituzione verrà usato anche il termine *Utente*.
- *Consumer*: un sito o applicazione che usa OAuth per accedere alle informazioni dell'utente nel *Service Provider*, su delega dell'*Utente* stesso.
- *Risorse Protette*: dati controllati dal *Service Provider*, alle quali 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 del utente e scambiato con un Access Token.
- *Access Token*: valore usato dal consumer per avere accesso alle risorse protette in delega del utente, invece di usare le credenziali dell'utente sul Service Provider.
- *Token Secre*t: valore segresso utilizzato dal consumer per stabilire la proprietà del token ricevuto.

2. Fasi del protocollo

Il protocollo di autenticazione OAuth si divide in due fasi. La prima parte definisce un processo, basato sulla redirezione da parte dell'user-agent2, per permettere agli utenti finali di autorizzare il client ad accedere alle loro risorse, attraverso l'autenticazione diretta con il server. Quest'ultimo fornirà al client, dopo aver effettuato l'autenticazione nel server, i tokens da usare nel metodo di autenticazione. 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 qualse vengono effettuate le richieste.

2.1 Autorizzazione del client

Come si è detto la prima parte del protocollo prevede che l'utente autorizzi il client. OAuth usa i token per rappresentare l'autorizzazione concessa al client dal resource owner. Tipicamente, i token sono emessi dal server su richiesta del'utente, dopo che quest'ultimo è stato autenticato, di solito attraverso l'inserimento di username e password.

Ci sono diversi modi con cui il server fornisce i token. Il metodo più comune è quello di utilizzare la redirezione HTTP e l'user-agent del resource owner. L'autorizzazione basata su questo metodo si struttura in tre fasi:

1. il client ottiene un set di credenziali temporanee dal server sotto forma di un identificatore e un shared-secret. Le credenziali temporanee sono usate per identificare la richiesta di accesso durante tutto il processo di autenticazione.
2. il resource owner autorizza il server ad accogliere la richiesta di accesso da parte del client identificato dalle credenziali temporanee.
3. il client usa le credenziali temporanee per richiedere un set di token dal server, che gli permetteranno di accedere alle risorse protette dell'utente.

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.
Per far si che il client possa compiere questi passi, il server ha bisogno di rendere pubbliche le URIs3 dei tre seguenti endpoints:

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

2.1.1 Temporary Credentials

La prima fase dell'autenticazione consiste nella richiesta, da parte del client, di un set di credenziali temporanee al server. Queste saranno usate per identificare univocamente la richiesta di accesso durante tutto il processo di autenticazione.
Il client per ottenere il set di credenziali temporanee dal server deve effettuare una richiesta HTTP "POST" all'endpoint per la richiesta delle credenziali temporanee costruendo una richiesta aggiungendo all' URI il seguente parametro (richiesto):
oauth_callback: un URI assoluto al quale il server redirezionera il'utente quando avrà completato lo step di autorizzazione.
Per fare la richiesta il client si autentica usando solamente le proprie credenziali.
Per esempio, il client effettua la seguente richiesta HTTPS:

```
POST /request_temp_credentials HTTP/1.1
Host: server.example.com
Authorization: OAuth realm="Example",
oauth_consumer_key="jd83jd92dhsh93js",
oauth_signature_method="PLAINTEXT",
oauth_callback="http%3A%2F%2Fclient.example.net%2Fcb%3Fx%3D1",
oauth_signature="ja893SD9%26"
```

A questo punto il server deve verificare la richiesta e se valida rispondere al client con un set di credenziali temporanee (sotto forma di identificatore 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_secre*t: shared secret delle credenzili temporanee.
- *oauth_callback_confirmed*: deve essere presente e settato a true. Questo parametro è usato per differenziarsi dalle versioni precedenti del protocollo.
-

Si noti che anche se i nomi dei parametri includono il termine token, queste credenziali NON sono inerenti al token, ma usate nei prossimi due passi in maniera simile alle credenziali di token.

- Esempio:

```
HTTP/1.1 200 OK
Content-Type: application/x-www-form-urlencoded
oauth_token=hdk48Djdsa&oauth_token_secret=xyz4992k83j47x0b&
oauth_callback_confirmed=true
```

2.1.2 Autorizzazione dell'utente

Una volta che il client ha ricevuto le credenziali temporanee, deve indirizzare l'utente al server per poter autorizzare la richiesta. Il client costruisce la URI per la richiesta aggiungendo dei parametri obbligatori all'endpoint del Resource Owner Endpoint:

- *oauth_token:* identificatore delle credenziali temporanee ottenuto al passo precedente nel parametro “oauth_token”.

Il client reindirizza l'utente a questa URI così costruita, attraverso una richiesta HTTP “GET”.
Per esempio, il client redirezione l'user-agent dell'utente per effettuare la seguente richiesta HTTPS:

```
GET /authorize_access?oauth_token=hdk48Djdsa HTTP/1.1
Host: server.example.com
```

Quando si chiede all'utente di autorizzare la richiesta di accesso, il server dovrebbe presentare all'utente le informazioni sul client che richiede accesso usando l'associazione delle credenziali temporanee con l'identità del client. Inoltre dovrebbe mostrare informazioni sul tipo di dati al quale il cliente vuole accedere.
Dopo aver ricevuto da parte dell'utente la decisione se autorizzare o meno il client, il server lo reidireziona alla URI di callback se questa è stata fornita nel parametro "oauth_callback" o in altra maniera se questa non è stata fornita.

Per assicurarsi che l'utente, che ha concesso l'autorizzazione, sia lo stesso che ritorna al client, per completare il processo di autenticazione, 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 i seguenti parametri alla callback URI:
oauth_token: identificatore delle credenziali temporanee ricevute dal client.
oauth_verifier: codice di verifica.
Esempio: *GET /cb?x=1&oauth_token=hdk48Djdsa&oauth_verifier=473f82d3 HTTP/1.1*
Host:client.example.net
Se il client non fornisce una URI di callback, il server dovrebbe visualizzare il valore del codice di verifica, e istruire l'utente su come informare manualmente il client che l'autorizzazione è completata.

2.1.3 Rilascio credenziali

L'ultimo passo per completare il processo di autorizzazione consiste nel rilascio da parte del server dei token autenticati. Il client li ottiene mediante una richiesta HTTP "POST" autenticata all'endpoint per le richieste dei token.
Il client costruisce una richiesta URI aggiungendo il seguente parametro (richiesto) , oltre ad altri (opzionali):
oauth_verifier: il codice di verifica ricevuto nello step precedente.
Al momento della richiesta, il client viene autenticato utilizzando sia le credenziali client sia le credenziali temporanee. Le credenziali temporanee sono utilizzate nella richiesta come sostitute per i token autenticati e vengono 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 (o canale sicuro equivalente) per garantire la sicurezza dei dati trasmessi. La stessa procedura deve essere fatta anche nel passo precedente.
Esempio:
POST /request_token HTTP/1.1
Host: server.example.com
Authorization: OAuth realm="Example",
oauth_consumer_key="jd83jd92dhsh93js",
oauth_token="hdk48Djdsa",
oauth_signature_method="PLAINTEXT",
oauth_verifier="473f82d3",
oauth_signature="ja893SD9%26xyz4992k83j47x0b"

Il server deve verificare la validità della richiesta, assicurandosi che l'utente abbia autorizzato il rilascio dei token autorizzati, e garantire che le credenziali temporanee non siano mai state usate prima e non siano scadute.
Il server deve anche 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 contiene i seguenti parametri richiesti:
oauth_token: identificativo del token.
oauth_token_secret: shared-secret del token.
Esempio:
HTTP/1.1 200 OK
Content-Type: application/x-www-form-urlencoded
oauth_token=j49ddk933skd9dks&oauth_token_secret=ll399dj47dskfjdk

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.

2.2. Richieste Autenticate

Una volta effettuati tutti i passi descritti nel capitolo precedente il client ha tutto cio che gli serve per eseguire richieste autenticate in nome del resource owner. Ad ogni richiesta il client dovrà presentare due set di credenziali: una che lo identifichi e l'altra che identifichi l'utente. Le credenziali del client sono in forma di coppia di chiavi RSA o una coppia identificatore/shared-secret.

2.2.1 Effettuare la richiesta

Gli steps che il client deve eseguire per effettuare una richiesta autenticata sono quattro:

1. Il client assegna i valori ad ognuno dei seguenti parametri obbligatori:
    - oauth_consumer_key: l'identificatore della 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 valor del time stamp. Il valore del timestamp deve essere un intero positivo. A meno di specifiche particolari dalla documentazione del server, il time stamp è espresso in numero di secondi dalle 00:00 del Gennaio 1970. Il parametro può essere omesso se si usa il metodo di firma "PLAINTEXT".
    - *oauth_nonce*: valore del nonce. Una nonce è 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 attack 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 omesso se si usa il metodo di firma "PLAINTEXT".
    - *oauth_version*: parametro opzionale. Se presente deve essere settato a 1.0. Fornisce la version del protocollo di autenticazione.
2. I parametri sono aggiunti alla richiesta. Ogni parametro non deve comparire più di una volta per richiesta.
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.
4. Il client manda la richiesta HTTP autenticata al server.

Esempio:
POST /request?b5=%3D%253D&a3=a&c%40=&a2=r%20b HTTP/1.1
Host: example.com

Content-Type: application/x-www-form-urlencoded
Authorization: OAuth realm="Example",
oauth_consumer_key="9djdj82h48djs9d2",
oauth_token="kkk9d7dh3k39sjv7",
oauth_signature_method="HMAC-SHA1",
oauth_timestamp="137131201",
oauth_nonce="7d8f3e4a",
oauth_signature="bYT5CMsGcbgUdFHObYMEfcx6bsw%3D"

Una volta che il server ha ricevuto la richiesta deve validarla: deve ricalcolare la firma indipendentemente e confrontarla con il paramatro "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.

2.2.2 Signature

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 "HMAC-SHA1", "RSA-SHA1" viene generata una stringa che è una concatenazione di diversi elementi della richiesta HTTP in un'unica stringa. 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.
Per esempio la richiesta HTTP seguente:

```
POST /request?b5=%3D%253D&a3=a&c%40=&a2=r%20b HTTP/1.1
  Host: example.com
  Content-Type: application/x-www-form-urlencoded
  Authorization: OAuth realm="Example",
              oauth_consumer_key="9djdj82h48djs9d2",
              oauth_token="kkk9d7dh3k39sjv7",
              oauth_signature_method="HMAC-SHA1",
              oauth_timestamp="137131201",
              oauth_nonce="7d8f3e4a",
              oauth_signature="bYT5CMsGcbgUdFHObYMEfcx6bsw%3D"
```

sarà rappresentata dalla seguente stringa:

```
POST&http%3A%2F%2Fexample.com%2Frequest&a2%3Dr%2520b%26a3%3D2%2520q
   %26a3%3Da%26b5%3D%253D%25253D%26c%2540%3D%26c2%3D%26oauth_consumer_
   key%3D9djdj82h48djs9d2%26oauth_nonce%3D7d8f3e4a%26oauth_signature_m
   ethod%3DHMAC-SHA1%26oauth_timestamp%3D137131201%26oauth_token%3Dkkk
   9d7dh3k39sjv7
```

2.2.3 HMAC-SHA1

Questo metodo usa l'algoritmo di firma HMAC-SHA1 così definito:

digest = HMAC-SHA1 (key, text)

dove il parametro text è il valore della stringa precedentemente costruita e key è la concatenazione della shared-secret fornito con le credenziali del client e lo shared-secret fornito con il token di autenticazione. Il digest è usato come valore del parametro “oauth_signature”.

2.2.4 RSA-SHA1

Questo metodo usa l'agoritmo di firma RSASSA-PKCS1-v1_5 e SHA1 come funzione hash. A differenza del metodo usa una coppia di chiavi pubblica e privata che il client deve aver scambiato con il server. La stringa è firmata con la chiave RSA privata del client:

S = RSA-SHA1(K, M)

dove K è la chiave privata del cliant e M è il valore della stringa precedentemente costruita. Mentre S è il risultato della cifrature ed è usato come valore del parametro “oauth_signature”.
Il server una volta ricevuta la richiesta verifica la firma in questo modo:

RSA-SHA1((n, e), M, S)

dove (n, e) è impostato con la chiave pubblica del client; M è la stringa da verificare mentre S è il parametro “oauth_signature” inviato dal client.

2.2.5 PLAINTEXT

Il metodo “PlainText” non utilizza nessun algoritmo di firma, ed è per questo motivo che deve essere utilizzato con un meccanismo, a livello di trasporto, come TLS o SSL per aumentare la sicurezza. Questo metodo non utilizza non crea, come si è visto precedentemente, la stringa per la firma e nemmeno i parametri timestamp e nonce.
Il parametro di protocollo “oauth_signature” verrà quindi settato con il seguente valore:

1. valore dello shared-secret del client.
2. Il valore “&”
3. valore del token secret

3. Considerazioni sulla sicurezza

Le maggiori fonti di rischio generalmente non si trovano nel cuore del protocollo stesso ma nelle policies e procedure che circondano il suo uso. Quindi è a carico del services provider che adotta OAuth come protocollo di autenticazione aggiungere degli accorgimenti per aumentare la sicurezza del protocollo stesso.

3.1 RSA-SHA1 Signature Method

La richiesta di autorizzazione fatta con la firma “RSA-SHA1” non usa una chiave segreta condivisa, questo significa che la sicurezza della richiesta si basa completamente sulla segretezza della chiave privata utilizzata dal client per firmare le richieste.

3.2. Confidenzialità della richiesta

Mentre questo protocollo fornisce un meccanismo per verificare l'integrità delle richieste, esso non fornisce garanzie sulla confidenzialità delle richieste. A meno che non siano state prese ulteriori precauzioni, i maliintenzionati avranno il pieno accesso al contenuto delle richieste. I server devono considerare attentamente i tipi di dati che possono essere inviati come parte di tali richieste, e dovrebbe impiegare meccanismi di sicurezza a livello di trasporto per proteggere le risorse sensibili.

3.3 Spoofing da parte di falsi Server

Questo protocollo non fa alcun tentativo per verificare l'autenticità del server. Una terza parte ostile potrebbe sfruttare questo per intercettare le richieste del client e rispondergli con risposte errate o ingannevoli. Quindi i service providers devono prendere in considerazione tali attacchi quando si sviluppa servizi che utilizzano questo protocollo, e dovrebbero richiedere la protezione a livello di trasporto per qualsiasi richiesta dove l'autenticità del server è un problema.

3.4 Secrecy of the Client Credentials

Il alcuni casi, l'applicazione client sarà sotto il controllo di parti potenzialmente non fidate. Per esempio, se il client è una applicazione desktop con il codice sorgente liberamente disponibile o un binario eseguibile, un aggressore potrebbe essere in grado di scaricarne una copia per l'analisi. In tali casi, i maleintenzionati saranno in grado di recuperare le credenziali del client. Di conseguenza, i server non dovrebbero utilizzare solo le credenziali del client per verificarne l'identità. Dove possibile, altri fattori come l'indirizzo IP.

3.5 Attacchi di Phishing

Un'ampia diffusione di questo protocollo e di altri simili può far si che gli utenti diventino avvezzi alla pratica di essere reindirizzati a siti Web dove viene chiesto di immettere la propria username e password. Se gli utenti non sono attenti a verificare l'autenticità di questi siti prima di fornire le loro credenziali, sarà possibile per i malintenzionati sfruttare questa pratica per rubare le password agli utenti.
I server dovrebbe tentare di educare gli utenti sui rischi che gli attacchi di phishing pongono fornendo dei meccanismi che rendano semplice per gli utenti verificare l'autenticità dei loro siti.

3.6 Scopo delle Richieste di Accesso

Di per se il protocollo non fornisce alcun metodo per la classificazione dei diritti di accesso concessi ad un client. Tuttavia, molte applicazioni richiedono una maggiore granularità sui diritti di accesso. Ad esempio, i server potrebbero voler rendere possibile l'accesso ad alcune protette risorse, ma non altre, o garantire solo un accesso limitato (come per esempio l'accesso in sola lettura) alle risorse protette.

Quando si implementa questo protocollo, i server dovrebbe prendere in considerazione i tipi di accesso alle risorse che l'utente può voler concedere al client, e dovrebbe fornire i meccanismi per farlo. Inoltre i server dovrebbero preoccuparsi che gli utenti capiscano i tipi di accesso che garantiscono al client, così come eventuali rischi connessi a ciò.

3.7 Entropy of Secrets

A meno che non si utilizzi un protocollo di sicurezza a livello di trasporto, gli intercettatori avranno un accesso completo alle richieste autenticate e quindi alle firme . Perciò saranno in grado di montare degli attacchi di forza bruta offline per recuperare le credenziali utilizzate. I server dovrebbe essere attenti a creare il segreto condiviso abbastanza a lungo, e abbastanza casuale da resistere a tali attacchi per almeno il periodo di tempo che i segreti condivisi dovranno essere validi. È altrettanto importante che il generatore di numeri pseudo-casuale (PRNG) utilizzato per generare questi segreti sia di qualità sufficientemente elevato. Molte implementazioni PRNG generano sequenze di numeri che possono sembrare casuale, ma che tuttavia presentano modelli o altre debolezze che rendono la crittanalisi o gli attacchi di forza bruta più facile.

3.8 Elaborazione automatica delle richieste di autorizzazione

I server potrebbe desiderare di elaborare automaticamente le le richieste di autorizzazione da clients che sono stati già precedentemente autorizzati dal proprietario della risorsa. Quando il proprietario della risorsa viene reindirizzato al server per concedere l'accesso, il server rileva che il proprietario della risorsa ha già concesso l'accesso a quel particolare client. Invece di chiedere conferma al proprietario della risorsa per l'approvazione, il server reindirizza automaticamente il'utente al client. Se le credenziali del client sono compromesse, l'elaborazione automatica crea ulteriori rischi per la sicurezza. Un utente malintenzionato può utilizzare le credenziali del client rubate per reindirizzare il proprietario della risorsa al server con una richiesta di autorizzazione. Il server concederà l'accesso ai dati del proprietario della risorsa senza l'approvazione esplicita del proprietario della risorsa, o anche la consapevolezza che è attacco. Se non viene implementata alcuna approvazione automatica, un utente malintenzionato deve utilizzare la “social engineering” per convincere il proprietario della risorsa di approvare l'accesso.
Server possono diminuire i rischi associati con l'elaborazione automatica, limitando l'operatività delle credenziali token ottenute attraverso le approvazioni automatizzate mentre quelle ottenute attraverso esplicito consenso del proprietario possono rimanere inalterate. A loro volta i clients possono mitigare i rischi associati con elaborazione automatica proteggendo lbene le lro credenziali.