Università degli Studi di Perugia

Facoltà di Scienze Matematiche, Fisiche e Naturali
Laurea Specialistica in Informatica

Corso di Sicurezza Informatica
Relazione Seminario:

OpenID

Studente
Pleqi Eno

Professore
Bistarelli Stefano

Anno Accademico 2008-2009

Indice

1 Introduction to OpenID

La gestione dell'identità è emersa come uno dei campi più importanti della tecnologia dell'informazione, soprattutto nella sicurezza dell'informazione. La gestione dell'identità è un meccanismo primario per il controllo degli accessi. Ogni utente che deve accedere a un conto bancario, siti web, eCommerce o una risorsa dell'azienda ha bisogno di un'identità per i controlli di accesso appropriati.

Ci sono molti modi per creare e gestire le identità digitali. In genere, le identità vengono gestite a livello di sistema operativo o a livello di applicazione anche se esistono altri luoghi per gestire le identità. Nella gestione a livello di sistema operativo tre sono gli ambienti più comunemente utilizzati: Unix/Linux, Microsoft Windows, and Mainframe.

- In UNIX/Linux, la gestione dell'identità viene fatta in diversi modi: **LDAP**, **NIS**, **RADIUS**, **Kerberos**, e altri meccanismi.
- Nel mondo Microsoft , **Active Directory** (AD) è il meccanismo più utilizzato.
- Ci sono molti meccanismi per la gestione dell'identità nei sistemi mainframe come ad esempio **RACF** (Resource Access Control Facility).

Vediamo le definizioni di questi meccanismi e protocolli:

- **LDAP,** Lightweight Directory Access Protocol, è un protocollo standard per l'interrogazione e la modifica dei servizi di directory.
- **NIS - Network Information Service** (originariamente chiamato Yellow Pages o YP) è un protocollo per servizi di directory client-server per la distribuzione dei dati di configurazione di sistema, come utenti e nomi host, tra computer su una rete.
- **Servizio di directory** è un programma o un insieme di programmi che provvedono ad organizzare e memorizzare informazioni su reti di computer e su risorse condivise disponibili tramite la rete. Il servizio di directory fornisce anche un controllo degli accessi sull'utilizzo delle risorse condivise in modo da favorire il lavoro dell'amministratore di sistema. I servizi di directory forniscono uno strato di astrazione tra le risorse e gli utenti.
- **RADIUS** (Remote Authentication Dial-In User Service) è un protocollo AAA (Authentication, Authorization, Accounting) utilizzato in applicazioni di accesso alle reti o di mobilità IP. RADIUS è attualmente lo standard de-facto per l'autenticazione remota, prevalendo sia nei sistemi nuovi che in quelli già esistenti.
- **Active Directory** è un insieme di servizi di rete meglio noti come directory service adottati dai sistemi operativi Microsoft a partire da Windows 2000 Server.
- **RACF** (Resource Access Control Facility) è un prodotto per la gestione della sicurezza creato dalla IBM per i sistemi operativi dei mainframe.

Oltre alle soluzioni basate sui sistemi operativi per la gestione dell'identità, un numero di prodotti commerciali e open source sono disponibili anche per questo scopo. I prodotti forniscono strutture per gestire l'identità degli utenti su più piattaforme e servizi, come Single Sign On (**SSO**), Cross Company Authentication (**CCA**), etc. Aziende come RSA, Novell, Sun e gli altri forniscono prodotti commerciali per la gestione identità sofisticati più piattaforme.

In genere questi sistemi funzionano molto bene in un ambiente chiuso dove tutte le applicazioni e sistemi sono gestiti da un'unica società (sono centralizzati). Con la popolarità dei sistemi basati sul web, gli utenti comuni hanno account su diversi siti web e questo è dove inizia il problema. Un utente deve ora creare identità per tutti i siti web e ricordare nome utente e passwords. Ovviamente, ciò ha creato alcune questioni non solo per gli utenti, ma anche visto da prospettive di sicurezza. Il problema più comune dell'utente che ha diversi account su diversi siti web è:

come gestire le molte identità in molti siti web con i quali l'utente deve interagire ?

OpenID è un protocollo aperto che permette ad una persona di utilizzare un URL come identità e utilizzare la stessa identità (URL) in più siti web che supportano OpenID. Applicazioni Web che supportano OpenID possono utilizzare l'URL di identità per l'autenticazione, autorizzazione e altri scopi. È un concetto relativamente nuovo che mette il controllo dell'identità nelle mani del suo proprietario, l'utente finale. Il proprietario dell'identità può decidere, e ha il controllo, su quale informazione dovrebbe essere presentata ad un'applicazione o sito web per scopi di autenticazione. Tra le altre cose, OpenID permette di:

- Effettuare il login alle applicazioni web senza inserire nome utente e password
- Abilitare i siti web a richiedere informazioni dall'utente, e l'utente a scegliere con quale card (con quale informazione) presentarsi presso il sito durante il processo di autenticazione / autorizzazione.
- La scelta delle "card" dipende sia dalla necessita del sito per le informazioni necessarie sia dal livello di importanza dei dati. Ad esempio per i conti bancari l'utente presenterà una card con più informazioni.
- Implementare un'alternativa a SSO per applicazioni multiple all'interno della stessa società.
- Implementare un meccanismo alternativo a CCA per l'autenticazione cross-company.
- Costi minori nell'implementazione e gestione.

2 Authentication and Authorization

Per stabilire l'identità di qualcuno viene utilizzata l'**autenticazione** e l'**autorizzazione** viene utilizzata per concedere o negare l'accesso a una risorsa dopo l'autenticazione. Tradizionalmente, nelle applicazioni web-based, autenticazione e autorizzazione sono basate molto sul sistema.

Autenticazione e autorizzazione si stanno comunque muovendo verso una logica basta sull'utente, verso metodi **user-centric**. Gli obiettivi dei metodi di autenticazione e autorizzazione basati sull'utente sono:

- Dare il controllo all'utente dei token di protezione che vengono inviati a un sito web per l'autenticazione e autorizzazione.
- Consentire ai siti web di richiedere i token o ulteriori parametri in base al livello del rischio di informazioni a cui si accede.
- Consente ai provider di identità (identity provider) di rilasciare le identità digitali anziché username e password. Le identità digitali possono contenere diversi set di token a seconda delle esigenze e delle circostanze.
- Semplificare il processo di autenticazione per gli utenti finali.

Per soddisfare questi obiettivi, sono stati sviluppati diversi tipi di sistemi. OpenID è uno dei leader nell'area open source.

2.1 What is An Identity?

Non è una questione nuova: è stata lì per secoli, molto prima che i computer venissero inventati. L'identità è qualcosa che viene utilizzata per distinguere qualcosa o qualcuno da altri. Se pensate sull'identità degli esseri umani, può essere il nome di una persona? Probabilmente no, perché da qualche parte in questo mondo, ci potrebbero essere molte altre persone con il vostro stesso nome. Può essere il social security number, il colore degli occhi, la lingua che parla una persona, il luogo che vivete o qualcos'altro?

Nella maggior parte dei casi, l'identità è una combinazione di fattori sopra menzionati. Vedremo come l'identità può essere stabilita e gestita nel migliore dei modi..

2.2 Authentication and Authorization

L'origine di autenticazione viene dal greco e ha il concetto di autentico o genuino. L'autenticazione è l'atto o il processo che autentica qualcosa o qualcuno. Il significato di dizionario per l'autenticazione è: stabilire qualcosa (o qualcuno) come un'entità valida.

In termini di protezione del computer, l'**autenticazione** è un processo con il quale un'entità (un utente, un'applicazione, un dispositivo, ecc.) accerta un'altra entità per ciò che pretende di essere. Il metodo di autenticazione più comunemente utilizzato è il nome utente e una password.

L'**autorizzazione** è il processo che in genere viene dopo l'autenticazione e viene utilizzato per concedere o negare l'accesso a un risorsa (resource computing). Così una volta che una persona o un dispositivo è stato autenticato, l'autorizzazione abilita il controllo dell'accesso a una risorsa per solo coloro che hanno un bisogno legittimo per ottenere tale l'accesso.

Ad esempio, potrebbe essere necessario il distintivo del dipendente per stabilire l'identità e entrare nell'edificio della società (autenticazione). Tuttavia, solo ad alcuni dipendenti è consentito andare nella sala stampa (autorizzazione).

Si noti che l'autenticazione e autorizzazione vanno di passo-passo. L'autorizzazione a diverse risorse può essere concessa a seconda di una moltitudine di fattori, tra cui:

- La sensibilità e l'importanza della risorsa svolge un ruolo importante. Ad esempio, a solo poche persone può essere concesso l'accesso all'informazione finanziaria della società.
- Il metodo di autenticazione, il che significa che se una persona effettua l'autenticazione con nome utente/password più un dispositivo biometrico possa ricevere un più elevato livello di accesso rispetto a una persona che effettua l'autenticazione solo con username e password.
- Seconda il giorno e ora, è possibile ottenere diversi livelli di accesso, soprattutto in luoghi come le istituzioni finanziarie, ad esempio la borsa.
- La posizione dell'entità che deve accedere alla risorsa è anche un fattore importante. Ad esempio, una connessione proveniente da un paese ostile disporrà di pochissimi diritti di accesso per una risorsa. Alcune società finanziarie impongono agli utenti di fornire informazioni aggiuntive al momento dell'autorizzazione, se la connessione è provenienti da fuori agli Stati Uniti.

Autenticazione e autorizzazione sono vincolate ad una categoria di informazioni molto più ampia, la gestione dell'identità ("identity management"), che sta diventando una parte molto importante della strategia IT globale. OpenID semplifica diversi problemi in alcuni settori della gestione di identità.

2.3 Authentication Methods

L'autenticazione per sistemi di computer viene eseguita utilizzando metodi diversi. Dipendente da una situazione particolare, la sensibilità dei dati o del sistema, un metodo o una combinazione di metodi verrà utilizzata per l'autenticazione. In questa sezione vengono descritti alcuni dei metodi di autenticazione più comunemente utilizzati.

2.3.1 Password Authentication

L'autenticazione basata sull'uso della password è il metodo più comunemente utilizzo in tutti i sistemi e applicazioni. L'autenticazione tramite password è stata utilizzata per accedere a sistemi operativi, applicazioni client - server, applicazioni desktop, come pure applicazioni basate sul web. Per molta gente, quando si parla di autenticazione, username e password è la prima cosa che viene in mente. L'autenticazione basata su password ha completato il suo compito molto bene nel tempo.

Tuttavia, alcune questioni di sicurezza sono nate con l'autenticazione basata su password. La questione più comune è semplicemente il numero di combinazioni nome utente/password che una persona deve ricordare. Maggior parte degli utenti di computer hanno un account presso il loro posto di lavoro, la loro casa, la loro istituzione finanziaria, i fornitori di assicurazione, email account, etc. È diventato quasi impossibile da ricordare tutte queste password. Di conseguenza, gli utenti tipici saranno:

- Quelli che scriveranno le loro password da qualche parte
- Quelli che utilizzeranno la stessa password per tutti gli account.

Entrambe le situazioni non sono buone in prospettiva di sicurezza. Se si scrivono le password su un documento, se qualcuno si impossessa di tale documento la sicurezza verrebbe a mancare. D'altro canto, se si utilizza la stessa password per tutti gli account di posta, tutti questi account saranno compromessi se la password venisse scoperta. Poiché la stessa password è memorizzata in molti luoghi diversi, la probabilità di sua divulgazione aumenta con l'aumentare del numero di posti in cui è memorizzata.

Inoltre, nel corso del tempo, gli aggressori hanno trovato molti modi per ottenere le password e gli strumenti di password cracking sono diventati molto sofisticati.

Gli attacchi di phishing hanno rivelato carenze nell'utilizzo delle password come mezzo unico per l'autenticazione.

2.3.2 PIN Authentication

Il **PIN** o Personal Identification Number è una stringa di numeri o una combinazione di numeri e lettere. In genere un PIN è minore in caratteri rispetto a una password. I PIN vengono utilizzati in molti scenari come carte ATM, autenticazioni basate su sistemi telefonici, noti anche come Interactive Voice Response (IVR) e così via. I PIN vengono utilizzati anche in dispositivi palmari dove è impraticabile l'utilizzo delle password lunghe.

I PIN sono considerati un meccanismo di autenticazione debole e possano essere facilmente scoperti dalla ricerca esaustiva. L'uso di PIN è un meccanismo di autenticazione ragionevole, finché esso è utilizzato in combinazione con qualcosa che avete o qualcosa che si è, noto anche come l'autenticazione a due fattori.

2.3.3 One Time Password (OTP) Authentication

OTP viene utilizzata una sola volta per evitare problemi con le password compromesse. Esistono diversi modi per generare una OTP. L'impiego di token elettronici è uno di questi metodi.

Una volta generata la password tale password rimane in uso per un po' di tempo.

Questi sono generalmente piccoli dispositivi. Questi dispositivi generano stringhe di numeri casuali, su intervalli di tempo specificati oppure quando un utente preme un pulsante sul dispositivo stesso.

L'utente dovrebbe quindi utilizzare il numero casuale per l'autenticazione presso un sistema. In genere questo numero casuale viene utilizzato in combinazione con un nome utente o con una password, oppure con una carta.

2.3.4 Smart Card Authentication

Le smart card sono solitamente analizzate o inserita in un sistema per scopi di autenticazione. Smart card vengono usate sia per sicurezza fisica, nonché per la sicurezza IT. Queste carte sono in genere abbastanza piccole (dimensioni di una carta di credito) affinché le persone possono portarle comodamente.

Molte volte smart card vengono utilizzate in combinazione con un PIN. Smart card hanno dei nastri magnetici e/o chip incorporato che sono in grado di fare una serie di cose. Vi sono questioni di interoperabilità con smart card.

2.3.5 Biometric Authentication

Nei sistemi con l'autenticazione biometrica vengono usate, le impronte digitali, la scansione della retina, o qualche altro meccanismo di identificazione dove viene utilizzato il corpo. I sistemi biometrici non sono molto scalabili e hanno anche problemi di affidabilità oltre ad essere costosi da installare e gestire. Metodi biometrici, se non utilizzati correttamente, possono anche causare ulteriori rischi per la privacy.

In un tipico sistema biometrico, alla persona verrà richiesto di immettere un pin oltre alla autenticazione biometrica. Alcuni portatili inoltre utilizzino l'autenticazione biometrica senza immettere un pin tale che per l'utente non è necessario effettuare il login per il laptop.

2.3.6 Certificate Based Authentication

Digital certificates (noto anche come certificati **x.509**) vengono utilizzati anche come meccanismi di autenticazione in molti scenari. SSH (Secure Shell) è un esempio dove i certificati sono utilizzati spesso per lo scopo di autenticazione. Molte aziende utilizzano certificati come secondo fattore per l'accesso remoto, ad esempio ad una VPN. L'autenticazione basata su certificati è considerato molto sicuro se un infrastruttura adeguata per la gestione del certificato è disponibile.

Tuttavia, le infrastrutture possono essere avere costi proibitivi per molte aziende. I certificati digitali possono essere revocati quando un dipendente lascia una società o quando un certificato è perso o rubato. I certificati digitali hanno anche un meccanismo di scadenza. Qualsiasi sistema che si basa su certificati digitali per scopi di autenticazione è di solito in grado di controllare la validità del certificato, scadenza o revoca. I provider del certificato mantengono solitamente un elenco, denominato *Certification Revocation List* (CRL), che tiene traccia dei certificati revocati.

2.3.7 USB Devices

Alcuni dispositivi USB gestiscono le informazioni di autenticazione, come un certificato X.509. Dispositivi USB sono facili da gestire rispetto alla smart card e possano essere utilizzati anche per portare con sé altri dati. Un altro vantaggio è che lo stesso dispositivo USB possa essere utilizzato per accedere a più credenziali di accesso.

Come altre tecnologie, ci sono molti rischi connessi con i dispositivi USB. Il rischio maggiore è che la gente perde molto spesso tali dispositivi USB e utilizzando questi dispositivi per il trasferimento dei file da un luogo a altro, questo pone un rischio significativo per un certificato che può venire rubato o compromesso.

2.4 Weak and Strong Authentication

Persone diverse definiscono l'autenticazione forte e debole in modi diversi. Più comunemente è inteso che se solo la combinazioni di username e password vengono utilizzati per l'autenticazione su un sistema, allora l'autenticazione viene chiamata **autenticazione debole**. Tuttavia, se si utilizza una combinazione di diversi metodi di autenticazione in un sistema, si chiama un metodo di **autenticazione forte**. L'utilizzo di appena un nome utente e password è debole perché essa può essere compromessa abbastanza facilmente rispetto a qualsiasi metodo di autenticazione forte.

Per l'autenticazione forte, è possibile utilizzare più metodi o fattori. Ad esempio, token OTP è un fattore e la password è un altro fattore. Fattori sono divisi in tre grandi categorie:

- ***Something you know***
- ***Something you have***
- ***Something you are***

L'autenticazione forte usa una combinazione dei metodi appartenenti a queste categorie.

2.4.1 Autenticazione a due fattori

Quando si utilizzano due fattori in combinazione, il sistema di autenticazione viene chiamato **autenticazione a due fattori**. Ciascuno dei due fattori dovrebbe comportare un successo per autenticare la persona. Ad esempio, se si utilizza nome utente/password e token OTP, sia la username/password e stringa di token OTP dovrebbero corrispondere per avere successo nell'autenticazione. In questo caso anche se qualcuno scoprisse la password, non potrà effettuare il login a meno che non ottiene anche il dispositivo per la gestione delle token.

L'autenticazione a due fattori è consigliabile per le concessioni finanziarie, mediche e dati sensibili del cliente.

Si noti inoltre che utilizzando due volte lo stesso fattore non si ottiene l'autenticazione a due fattori. Ad esempio, utilizzando due password, anche se diverse, non rende l'autenticazione a due fattori. Per l'autenticazione a due fattori, si devono usare due fattori da due delle tre categorie elencate precedentemente in questo capitolo (cosa so,cosa possiedo o cosa sono).

2.4.2 Single Sign-On (SSO)

Anche per single sign-on SSO è difficile dare una definizione globalmente accettabile. Una definizione valida:

> **è un meccanismo secondo il quale è necessario effettuare il login una volta per ottenere l'accesso a più risorse.**

Così ad esempio, vi possono essere molte applicazioni nella rete aziendale. Tuttavia, quando si logga su un'applicazione e quindi si sposta su un altra , non è necessario autenticarsi nuovamente. Le credenziali sono consegnate a tutte le applicazioni che prendono parte in un sistema SSO e tutto questo avviene in background. A seconda del meccanismo di background utilizzato per SSO, alcuni dei meccanismi non sono considerati come veri SSO. In tali situazioni, viene anche chiamato "**accesso semplificato**" invece di un vero SSO.

SSO risolve una serie di problemi con username e password. Elimina la necessità di più username e password che sono difficili da ricordare. Con SSO, è possibile implementare migliori controlli per la difficoltà della password cosi che gli utenti devono scegliere una password difficile da trovare.

Tuttavia, SSO è come la chiave del castello, e se si perde, qualcuno può accedere a tutte le applicazioni e sistemi disponibili. È più prudente utilizzare SSO con l'autenticazione a due fattori in modo tale che anche se si perde una chiave non si può accedere a tutte le stanze del castello, cosi il rischio associato al sistema SSO è ridotto.

Molti sistemi sono sviluppati per SSO da diverse società. Kerberos è un sistema di tre parti che funziona sulla base di credenziali o biglietti. Inizialmente quando un utente accede al sistema, all'utente è concesso un biglietto. Quando lo stesso utente ha bisogno di accedere a un'altra risorsa della rete, un altro biglietto viene generato utilizzando il biglietto esistente.

In genere SSO viene utilizzato all'interno di una società e diventa inutile quando è necessaria l'autenticazione attraverso diverse società. Questo è perché un utente potrebbe avere uno username e password degli account di una società e un'altra serie di username e password per un'altra società. OpenID può essere utilizzato anche per SSO.

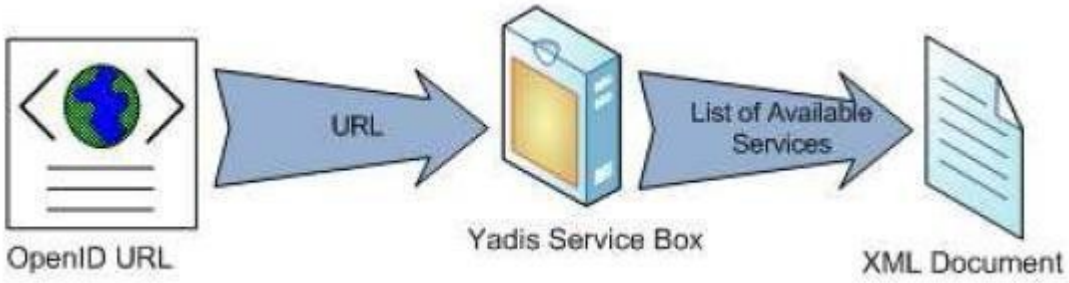
2.5 New Authentication Mechanisms - Yadis

Yadis fornisce un meccanismo per scoprire i servizi disponibili in un determinato URL.OpenID. È un semplice protocollo basato su XML e consente a un consumer di scoprire l'autenticazione nonché altri servizi forniti da un URL di identità.

Dato un URL di identità e nessun altra informazione, come facciamo a sappere quale protocollo deve essere utilizzato per autenticare l'utente? Yadis è un sistema di scoperta del servizio che consente al consumer (il Relayin Party) determinare automaticamente, senza l'intervento dell'utente finale, il protocollo più appropriato da utilizzare.

Yadis fornisce il primo passo per qualsiasi servizio che utilizza gli identificatori per l'autenticazione, autorizzazione, scambio di dati , e altro. Lo scopo di Yadis e quello di permettere al consumer (RP) di ottenere un file XRD (eXtensible Resource Descriptor). Per ottenere questo file il consumer effettua una richiesta HTTP. In risposta a tale richiesta il consumer (RP) può ottenere:

1. un documento XRD
2. un'url che contiene un file XRD valido.



La figura sotto mostra la “scatola nera” di Yadis che prende un URL come input e fornisce un documento XML come output..

3 OpenID Protocol and Messages

OpenID è un sistema che consente a un utente di utilizzare una Uniform Resource Identifier o URI (come l'URL del sito web) per scopi di autenticazione. Questo URI viene utilizzato come username o identità per un persona e viene anche chiamato "**identifier**".

3.1 Terminologia OpenID

End User

L'utente finale è l'utente reale che utilizza il sistema OpenID per effettuare il login su siti web diversi utilizzando la propria credenziale memorizzate presso il provider di identità.

Consumer or Relying Party (RP)

I consumatori "c*onsumer"* sono i siti web dove si effettuerà il login utilizzando OpenID. Si chiamano "consumatori" perché che consumano, usano, le credenziali OpenID fornite dal provider di identità. Tutti i siti web che supportano il login con OpenID sono consumatori detti anche Relying Party or RP (Parte Fidata).

Identifier

Identificatore "*Identifier"*è l'URL che identifica l'identità digitale dell'End User.

Identity Provider or IdP (OP)

Provider di identità "Identity Provider or IdP" è l'host dove vengono memorizzate le credenziali dell'utente. L'URI OpenID indica il provider di identità. Durante il processo di autenticazione, il consumer convaliderà un ID scambiando alcuni messaggi con il provider di identità. A volte viene anche chiamato come OpenID Server, OpenID Provider o semplicemente OP.

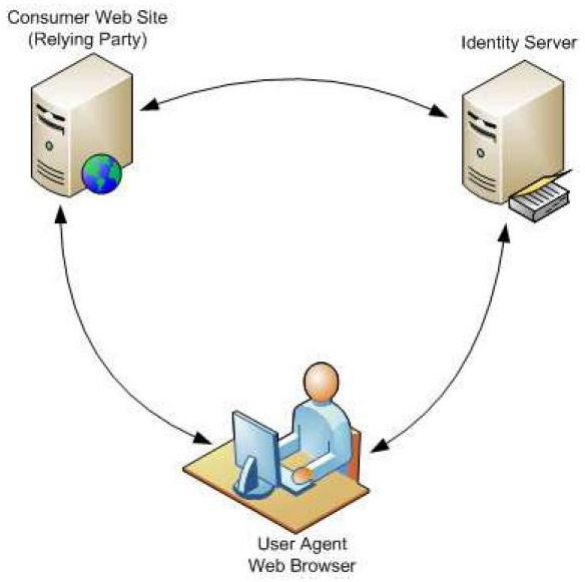
User Agent

Il "User Agent" non è altro che il browser dell'utente, con il quale comunica direttamente.

3.2 Comunicazione tra componenti in un sistema OpenID

Esistono tre principali componenti in qualsiasi sistema OpenID: "**consumer**", "**identity provider**", "**user agent**". Questi componenti interagiscono tra loro durante il processo di autenticazione:

- Il consumer, interagisce con l'identity provider e con lo user agent. L'utente finale tenterà di accedere al sito web del consumer tramite OpenID. Durante il processo di autenticazione, il consumer invierà alcuni messaggi al identity provider direttamente, nonché tramite lo user agent con l'aiuto di HTTP redirect messages.
- L'identity provider è il server OpenID che contiene le credenziali dell'utente finale. L'identity provider convaliderà il proprietario del URL per il consumer utilizzando due meccanismi fondamentali che vedremo in seguito.
- L'utente finale interagisce con il consumer e l'identity provider mediante lo user agent.



Durante il processo di autenticazione, il browser agisce come "**middle man**" tra il provider di identità e il sito web del consumatore per alcuni messaggi. In genere, un consumer interagisce con il browser web, cosi come con il provider di identità durante il processo di autenticazione. Tuttavia in alcuni casi, il consumer può utilizzare delle chiavi memorizzate nella cache per autenticare un utente senza alcuna comunicazione diretta con il provider di identità.

3.3 Direct and Indirect Communication

Esistono due metodi di comunicazione base tra le entità diverse in un sistema OpenID:

- comunicazione diretta e
- comunicazione indiretta.

Con il meccanismo della comunicazione diretta, le due entità dialogano direttamente tra loro tramite il protocollo HTTP. Il metodo POST HTTP viene utilizzato per la comunicazione diretta.

Con la comunicazioni indiretta, le due entità dialogano tra loro tramite una terza entità. Questa terza entità è in genere il browser web. La comunicazione indiretta può avvenire tramite HTTP Redirect o tramite HTML Form redirection.

3.4 OpenID Modes of Operation

OpenID ha due modalità principali di operare:

- la modalità Dumb
- la modalità Smart.

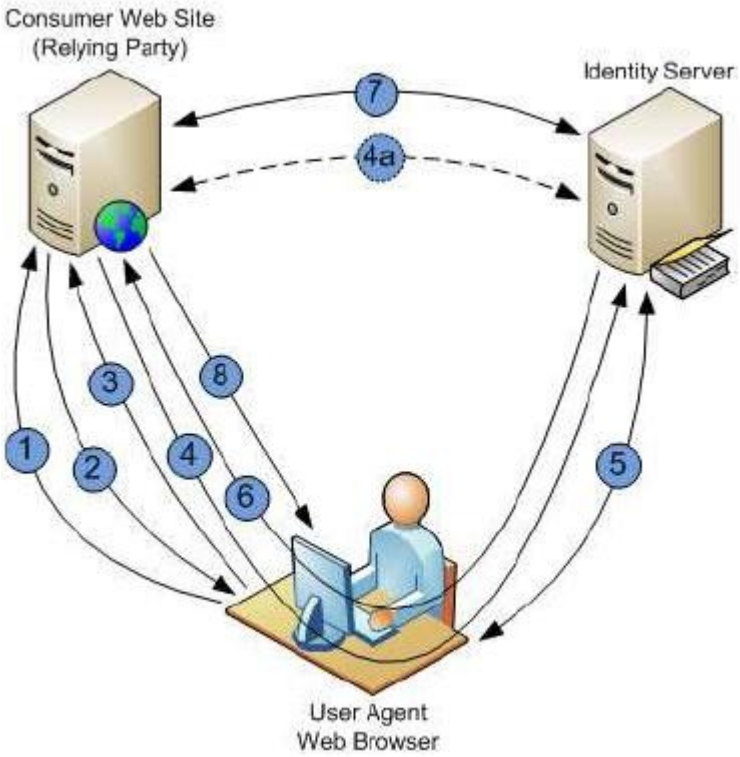
Queste modalità si basano nell'intelligenza del consumer. In modalità **dumb**, come indica il nome, il consumatore non è cosi smart e deve eseguire alcuni passaggi aggiuntivi ogni volta che un utente accede. In modalità **smart**, il consumer mantiene informazioni sullo stato e mette nella cache le chiavi condivise per un utilizzo futuro. In questa sezione, si vedranno informazioni più dettagliate sul funzionamento di queste due modalità.

3.4.1 Dumb Mode Communications Flow

In modalità Dumb, i consumer non mantengono lo stato della connessione, quindi qualsiasi informazione che è stata utilizzata in un precedente login, non potrà essere più utilizzata. Ogni volta che un End User accede a un Dumb Consumer web site, viene ripetuto lo stesso processo. Prendiamo come esempio

- Identity Provider: myopenid
- Relying Party: livejournal

In seguto vediamo step – by – step il processo di autenticazione e login quando viene effettuata la comunicazione fra: il consumer, lo user agent e l'identity provider. La figura sotto da una presentazione grafica di tali passaggi:



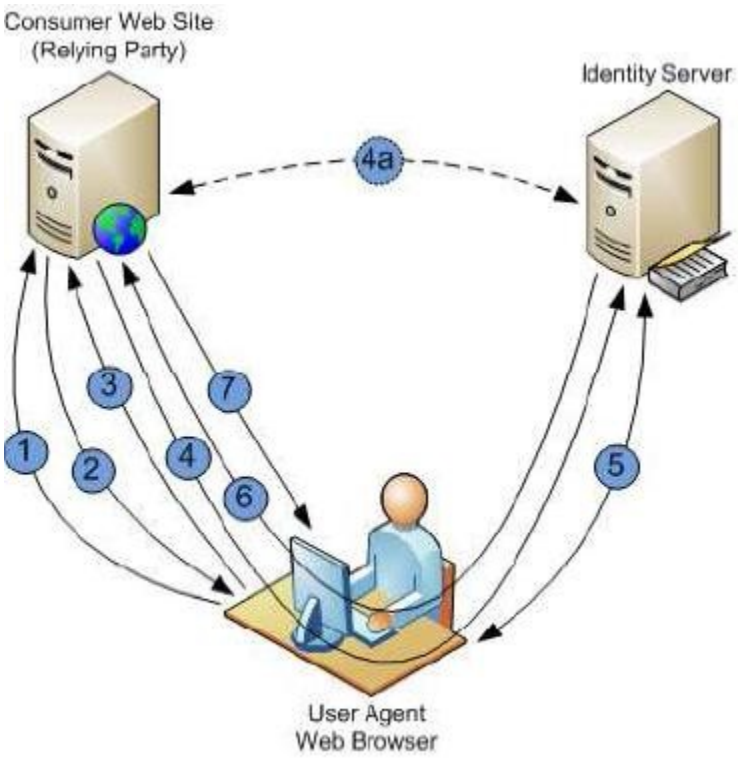
1. accediamo alla pagina web del consumer: http://www.livejournal.com/
2. inseriamo l'**url** dell'identità ed effettuaiamo il login : nome_utente.myopenid.com

   Da notare che non è necessario aggiungere "http:// " all'inizio dell'URL. Le specifiche dell'OpenID richiedono che tutti i consumer dovrebbero essere abbastanza intelligenti a comprendere un URL in formati diversi.
3. Il sito web del consumer (livejournal.com) pulirà l'identifier URL e recupererà le informazioni da tale URL. In OpenID 2.0, un altro protocollo XML, conosciuto come **Yadis**, può essere utilizzato in questa fase di rilevamento del servizio.
4. Dopo il recupero della pagina, il consumer (livejournal.com) analizzerà e determinerà la posizione del provider di identità (OpenID Server). Questa informazione è incorporata all'interno della pagina web HTML. Questo processo di analisi viene anche chiamato "discovery". Dopo l'analisi, il consumer (livejournal.com) reindirizzerà il browser al Identity Provider per ottenere le informazioni necessarie per l'autenticazione. Questo avviene utilizzando il metodo HTTP GET.

   4.a Facoltativamente, il consumatore può stabilire una connessione a questo punto con il provider di identità e condividere una chiave segreta per un'ulteriore comunicazione. Questo è illustrato con una linea tratteggiata nella figura e contrassegnato come passo 4 a.
5. Se l'End User non è già loggato presso l'Identity Provider, l'Identity Provider chiederà all'End User di effettuare il login. Questa parte è esterna al protocollo OpenId ed è lascia all'Identity Provider la facoltà di decidere come autenticare l'End User. Se l'utente è già loggato nell'Identiy Provider questa parte viene saltata.
6. L'identity provider (myopenid.com) ritornerà l'informazione necessaria con la sua firma al Consumer via browser redirect. Questa informazione contenente il risultato dell'autenticazione sia che è andato a buon fine (success) oppure no (failure). Il metodo HTTP GET è utilizzato in questo step. Da notare che c'è una comunicazione indiretta fra l'Identity Provider e il Consumer.
7. Durante l'autenticazione con successo, il consumer (livejournal.com) dovrà stabilire una connessione diretta con l'Identity Provider (myopenid.com), preferibilmente in tramite una sessione SSL sicura. Il Consumer potrà richiedere le informazioni di autenticazione direttamente dall'Identity Provider e confrontarle con le informazioni di asserzione ricevuta tramite User Agent (browser web). Questo è un doppio controllo per la validità dell'affermazione nel caso in qui uno User Agent (o un utente malintenzionato) sta cercando di imbrogliare.
8. Se tale confronto va a buon fine allora l'End User verrà loggato nel sistema, altrimenti il login fallirà.

3.4.2 Smart Mode

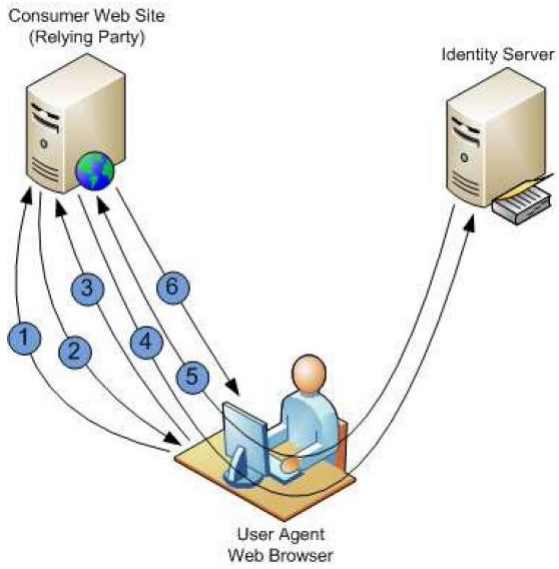
La modalità "Smart" per l'autenticazione è simile alla modalità "Dumb" con l'unica eccezione allo step 7. In questo caso il consumer ha già la chiave segreta (dal passo 4a) e può decriptare e verificare l'affermazione dal step 6 e determinare se l'Identity Provider ha già autenticato l'End User. Da notare che lo step 4a viene effettuato una volta ogni tanto, quando l'End User svuota la cache oppure quando effettua la richiesta di autenticazione la prima volta.



1. L'end user visita la pagina web del counsumer
2. La pagina web del consumer si presenta con una form nella quale l'utente inserisce il suo identity URL e clicka l'invio.
3. Il consumer pulisce l'identifier url e prende la pagina puntata dall'url.
4. Dopo aver preso la pagina dall'url dell'utente, la parsa, e determina la posizione dell'identity provider. Dopo che ha effettuato il parsing della pagina, il consumer, effettuerà un redirect del browser verso l'identity provider per ottenere l'affermazione della validità dell'indetifier che l'utente ha inserito. In questo step, il consumer può inviare una messaggio di “association request” all'identity provider e scambiare con lui una chiave condivisa (step 4.a).
5. Se l'utente non è loggato presso l'identity provider, allora l'identity provider può richiedere all'end user di effettuare l'autenticazione.
6. L'identity provider restituirà le informazioni di affermazione con la firma per il consumatori tramite browser redirect. Tale affermazione rappresenterà sia un successo di autenticazione o il suo fallimento.
7. Dopo un'affermazione di successo, il consumer verifica l'affermazione utilizzando la chiave condivisa nella cache. Se c'è una corrispondenza nel passaggio precedente, l'utente finale verrà login al sito web. Altrimenti il login fallirà.

In molti casi quando si hanno già stabilite le credenziali, il provider di identità non richiederà nuovamente per il login. Ci sono molte tecniche che possano essere utilizzate dal provider di identità per raggiungere questo obiettivo, tra cui active browser session e i cookie. In tale scenario, e supponendo che il consumer ha già memorizzato nella cache la chiave condivisa con l'dentity provider la comunicazione diventerà molto semplice e l'utente finale potrà effettuare il login al sito web del consumer senza alcuna sessione interattiva con il provider di identità.

Ciò è mostrato in figura:



Come si vede in figura le comunicazioni con l'identity provider sono trasparenti all'utente, e tutte le operazioni succedono dietro le scena via browser redirect.

In questo caso, i passaggi saranno i seguenti. Si noti che l'utente finale potrà immediatamente accedere al sito web dopo aver immesso l'URL di identificatore:

1. Visita del sito web del consumer
2. Il sito web presenta la pagina dove si inserisce l'identifier url.
3. Il consumer pulisce l'identifier url e prende la pagina puntata dall'url.
4. Dopo aver preso la pagina dall'url dell'utente, la parsa, e determina la posizione dell'identity provider.
5. L'identity provider restituirà le informazioni di affermazione con la firma per il consumatori tramite browser redirect. Tale affermazione rappresenterà sia un successo di autenticazione o il suo fallimento.
6. Dopo un'affermazione di successo, il consumer verifica l'affermazione utilizzando la chiave condivisa nella cache. Se c'è una corrispondenza nel passaggio precedente, l'utente finale verrà login al sito web. Altrimenti il login fallirà.

3.5 OpenID Identity URL Page

Diamo un occhiata all'informazione presente nella pagina dell'identità OpenId. Questa pagina può essere ospitata preso un qualsiasi server web (non necessariamente al server web dell'identity provider). Basta creare una pagina HTML con le specifiche informazioni e metterla in un sito web. L'url presso questo documento sarà il nostro identificatore. Il documento HTML avrà l'informazione riguardante il server OpenId (il provider) nella sessione HEAD della pagina.

```
<link rel="openid.server" href=https://myopenid.com/ >
```

Si noti che nella riga sopra,"**openid.server**" è una parola chiave e dovrebbe apparire esattamente come indicato sopra. La parte "href" può variare a seconda dell'identity provider che si utilizza. Si noti inoltre che l'URL per il provider di identità possa iniziare con HTTP o HTTPS a seconda della configurazione dell'identity provider.

Vediamo un esempio di una pagina HTML puntata dall'url dell'identità:

```
<html>
	<head>
		<link rel="openid.server" href="http://idp.conformix.com/index.php/serve">
		<link rel="openid.delegate" href="http://idp.conformix.com/?user=test">
	</head>
<body>
	<h3>OpenID Identity Page</h3>
</body>
</html>
```

In questo caso le seguenti linee indicano il percorso del server OpenID che il consumer contatterà per l'autenticazione. Usando questo meccanismo si può separare tale URL dal URL dell'identificatore.

```
<link rel="openid.server" href="http://idp.conformix.com/index.php/serve">
```

La riga seguente mostra l'URL dell'OpenID. Da notare che la parola chiave "***delegate***" viene usata quando l'URL dell'identificatore è ospitato in un'altra macchina diversa dal server.

```
<link rel="openid.delegate" href="http://idp.conformix.com/?user=openidbook">
```

Se il consumer e l'OpenID Server supportano le specificazioni OpenID 2.0, può essere usato anche un documento XRD al posto del documento HTML. Un tipico documento XRD si presenta come segue:

```
<?xml version="1.0" encoding="UTF-8"?>
	<xrds:XRDS
		xmlns:xrds="xri://$xrds"
		xmlns:openid="http://openid.net/xmlns/1.0"
		xmlns="xri://$xrd*($v*2.0)">
<XRD>
	<Service>
		<Type>http://openid.net/signon/1.1</Type>
		<Type>http://openid.net/sreg/1.0</Type>
		<URI>http://idp.conformix.com/index.php/serve</URI>
		<openid:Delegate>
			http://idp.conformix.com/?user=openidbook</openid:Delegate>
	</Service>
</XRD>
</xrds:XRDS>
```

3.6 OpenID Messages

I componenti OpenID scambiano diversi messaggi durante il processo di autenticazione. I formati di questi messaggi sono ben definiti e i Consumers e Identity Providers devono rispettare questi formati per una comunicazione corretta.

Ogni messaggio ha una combinazione di richiesta e risposta e la richiesta e risposta possono avere diversi set di parametri. Vedremo i parametri di richiesta e risposta separatamente.

Come abbiamo visto in precedenza, il Consumer e l'Identity Provider possono usare la comunicazione diretta o indiretta. Per la comunicazione diretta viene usato il metodo "HTTP POST" mentre per quella indiretta "HTTP GET".

Ci sono quattro tipi principali di messaggi usati in un sistema OpenID:

1. *associate* message
2. *checkid_immediate* message
3. *checkid_setup* message
4. *check_authentication* message

3.6.1 The associate Request Message

L'"associate message" è inviato dal Consumer (Relying Party o RP) all'Identity Provider (o OP). Lo scopo principale di questo messaggio è quello di stabilire un "segreto condiviso" tra queste due entità. Il sito web del Consumer può inviare questo messaggio al Identity Provider in qualsiasi momento è richiesto (quando si svuota la cache oppure nella comunicazione smart).

Siccome questa è una comunicazione diretta tra il Consumer e l'Identity Provider, il metodo "HTTP POST" sarà usato per i messaggi di tipo "associate message". Mentre si avvia la richiesta di "associate", il Consumer invierà una serie di parametri insieme alla richiesta. Vediamo una lista di tali parametri che possono essere inviati:

- **openid.ns** : Opzionale, serve per specificare la versione del protocollo OpenID (1.1 – 2.0)
- **openid.mode** : indica il tipo di messaggio e serve per distinguere quale messaggio è stato inviato o ricevuto. Il valore per questo parametro è "associate" per un messaggio di "associate request".
- **openid.assoc_type** : questo parametro viene usato per indicare il tipo di algoritmo utilizzato per la firma del messaggio. OpenID utilizza i seguenti algoritmi per la firma:
  - "HMAC-SHA1"
  - "HMAC-SHA256"
- **openid.session_type** : è utilizzato per indicare il tipo di crittografia utilizzato nel messaggio per crittografare la chiave MAC (Message Authentication Code). Un MAC è breve codice che viene calcolata utilizzando una chiave segreta e il messaggio in input. Un algoritmo MAC prende la chiave privata e il messaggio come input e fornisce il codice MAC come output. Hash Message Authentication Code o HMAC è un tipo di MAC.

  Nei messaggi di OpenID, diversi tipi di crittografia vengono utilizzati per crittografare MAC riportati di seguito:

  - "DH-SHA1" per Diffie-Hellman SHA1

- “DH-SHA256” per Diffie-Hellman SHA256
- Il valore “no-encryption” per inviare il MAC in chiaro, anche se è fortemente raccomandato crittografarlo. Se si usa una sessione di connessione SSL tra il Consumer e l’Identity Provider si può optare a non crittografare il MAC poiche il trasport layer si occupa della crittografia.

- **openid.dh_modulus** - **openid.dh_gen** - **openid.dh_consumer_public**: questi tre parametri vengono usati nel caso in cui viene utilizzato DH-SHA1 o DH-SHA256 per openid.session_type.

Vediamo una tipica richiesta di “**associate Request Message**”

```
openid.mode=associate&openid.assoc_type=HMAC-SHA1&openid.session_type=DHSHA1&
openid.dh_consumer_public=KC6IpA00A6SlCikafFSlrTGql9H8+de6GFi5YLKz4p
yDxUMS5Z8pMOm/Ptr1gFmCcgAXjFbuxS73ZutDTFJYpADoIntFVrah9eaezMcw6SDR24cnFjN
c14xq0zGt3QcRLXaNTRVKfMW8evDAmLCrvEhU5c7B3eqmk+bMMrbQpcE=&openid.dh_modul
us=ANz5OguIOXLsDhmYmsWizjEOHTdxfo2Vcbt2I3MYZuYe91ouJ4mLBX+YkcLiemOcPym2CB
RYHNOyyjmG0mg3BVd9RcLn5S3IHHoXGHblzqdLFEi/368Ygo79JRnxTkXjgmY0rxlJ5bU1zIK
aSDuKdiI+XUkKJX8Fvf8W8vsixYOr&openid.dh_gen=Ag==
```

3.6.2 The associate Response Message

Il messaggio “associate response” viene inviato dall’Identity Provider al Consumer (o RP). Il messaggio può mostrare un'associazione con successo o fallimento. Nel caso in cui l’associazione è andata a buon fine, il messaggio conterrà un “handle” e la vita di quel “handle” in secondi. Nel caso in cui l’associazione è andata male verrà restituito un messaggio di errore. Lista dei parametri:

- **openid.ns**
- **openid.assoc_handle**: è una stringa ASCII con lunghezza massima di 255 caratteri. Tale “association handle” può essere utilizzata nei messaggi successivi. In genere questo “handle” dura per un po' di tempo. Viene utilizzata per determinare quale chiave dovrebbe essere riutilizzata per la crittografia/decrittografia.
- **openid.session_type**: è uguale al caso di request, se l'associazione fallisce per qualsiasi motivo, il valore di questo parametro sarà “unsuccessful response”. Possono esserci diversi motivi per un'associazione non riuscita. Ad esempio, se il Consumer chiede di utilizzare SHA256 e l’Identity Provider può supportare solo SHA1, l'associazione avrà esito negativo.
- **openid.assoc_type:** è uguale al caso di request, “unsuccessful response” nel caso di un fallimento
- **openid.expires:in**: è il tempo in secondi dopo di ché l’associazione scade e il Consumer (RP) deve chiedere una nuova associazione (ttl della handle).
- **openid.mac_key**: parametro che viene usato solo nel caso in cui il valore per “openid.session_type” era “no-encryption”. Il valore per questo parametro è una chiave MAC codificata e in base 64.
- **openid.server_public**: è la chiave pubblica dell’Identity Provider. Questo parametro viene utilizzato se è stato utilizzato l’algoritmo di Diffie-Hellman nella richiesta.
- **openid.enc_mac_key**: è la chiave MAC codificata. Questo parametro viene utilizzato se è stato utilizzato l’algoritmo di Diffie-Hellman nella richiesta.

In caso di insuccesso, vengono restituiti i parametri “error” e “error_code”. La risposta può avere anche altri parametri facoltativi, ma i parametri di errore ed error_code sono importanti.

Esempio:

```
assoc_handle:{HMAC-SHA1}{4607344a}{oDFF0g==}
assoc_type:HMAC-SHA1
dh_server_public:AIPkx6xJ3b1Wnr1olWL7suoZnABDc+lJRR9DeNIBolGXQX3W2e+4udY2
p+dUcF5jKE6uoZuXLVPbimHbndBOYhUDUfkKaAjQtVvONerAjd5RHyt2i2AoYrkjD26traC4j
zg7NukZlmrRjfPRg4q3gwW+EZEXvz+ba9JnQfsXx+iH
enc_mac_key:UtQHBswQimAZAp4s/9sfSQSpuq0=
expires_in:1209600
session_type:DH-SHA1
```

3.6.3 The checkid_setup and checkid_immediate Request Messages

I messaggi **checkid_immediate** e **checkid_setup** vengono utilizzati per ottenere informazioni di affermazione dal server OpenID. Questi messaggi vengono avviati dal Consumer. Per questi messaggi viene utilizzata la comunicazione indiretta, che significa che il consumer utilizzerà il metodo GET HTTP (invece di HTTP POST) per l'invio e ricezione di tali messaggi. Questo significa che questi messaggi passeranno per lo User Agent (il browser web). I seguenti parametri vengono usati con i messaggi “checkid_setup request”.

- **openid.ns**
- **openid.mode:** che ha come valore “checkid_setup”
- **openid.claimed_id:** parametro opzionale visualizza l’identificatore che l’utente dichiara di possedere, ma che non è ancora verificato.
- **openid.identity:** opzionale
- **openid.assoc_handle**: opzionale e se presente indica che è già stata stabilita una associazione fra il Consumer e l’Identity Provider.
- **openid.return_to**: parametro usato per informare il server OpenID dell’URL dove deve reindirizzare il browser dopo aver processato la richiesta. Il server OpenID userà questo URL per inviare la risposta al Consumer.
- **openid.realm**: opzionale è un’URL che il server OpenID usa per identificare il Consumer. Questo parametro può contenere caratteri speciali come ad esempio “*”.

Da notare inoltre che il messaggio di richiesta raggiunge il server OpenID in due passaggi. Nel primo passaggio, il metodo HTTP redirect viene utilizzato dal Consumer per il browser web. Quindi nel passaggio successivo, il browser invia la richiesta GET HTTP al server di OpenID.

Esempio di un messaggio checkid_setup request:

```
GET /index.php/serve?openid.assoc_handle={HMACSHA1}{
46071e25}{Tt8MwQ==}&openid.identity=http://idp.conformix.com/?user=
openidbook&openid.mode=checkid_setup&openid.return_to=http://consumer.con
formix.com:80/finish_auth.php?nonce=nC5sKquX&openid.sreg.optional=email&o
penid.trust_root=http://consumer.conformix.com:80/ HTTP/1.1
```

3.6.4 The checkid_setup and checkid_immediate Response Messages

Una volta ricevuto il messaggio “checked_setup” il server OpenID effettua alcune operazioni ed invia la risposta indietro al Consumer via browser. Opzionalmente il server OpenID può chiedere all’End User di autenticarsi preso questo server per essere sicuro che solo il proprietario dell’Identifier Url può autorizzare l’accesso all’Identifier. Questo step opzionale può presentarsi con una pagina di login, ma come detto in precedenza l’implementazione di questa parte riguarda il server OpenID e non è inclusa nelle specifiche di OpenID.Nel messaggio della risposta, il server OpenID invierà alcuni parametri indietro al Consumer. Alcuni di questi parametri sono opzionali e alcuni necessari:

- **openid.ns**
- **openid.mode:** che avrà come valore “id_res”. Nel caso in cui l’associazione fallisce il valore per questo parametro sarà:
  - “setup_needed” nel caso in cui la richiesta fosse “checkid_immediate”
  - “cancel” nel caso in cui la richiesta fosse “checkid_setup”
- **openid.op_endpoint:** visualizza l’URL del server OpenID
- **openid.claimed_id**
- **openid.identity**
- openid.assoc_handle: rappresenta l'handle che è stato utilizzato per firmare questo messaggio. Il consumer utilizzerà questo handle per finalità di verifica. Questo handle può coincidere con quello che ha inviato il Consumer con il messaggio di richiesta (nel caso esiste già un'associazione). Può essere diverso da quello originale che il Consumer ha inviato con la richiesta se il server OpenID non riconosce l'handle originale. In tal caso, il server deve mantenere un record del nuovo handle di modo che il Consumer possa utilizzarlo a scopo di verifica (utilizzando il messaggio check_authentication discusso tra breve).
- **openid.return_to**: è identico allo stesso parametro inviato dal Consumer con il messaggio di request.
- **openid.response_nonce**: parametro utilizzato per evitare relay attacks, è univoco per ogni messaggio. La lunghezza massima di una **nonce** è di 255 caratteri ed è costituito dal server timestamp e altri caratteri ASCII per renderlo univoco. Il Consumer può respingere come associazione se il timestamp è troppo lontano dal tempo corrente.
- **openid.invalidate_handle**: è utilizzato per indicare se l'handle collegato alla richiesta era valida o no. Se l'handle era valido, questo parametro è facoltativo. In caso contrario, l'handle non valido debba essere collegato con questo parametro. Questo aiuterà il Consumer a rimuovere tale handle dai suoi record.
- **openid.signed**: contiene un elenco di parametri che sono firmati. L'elenco è separato da virgola.

Come con il messaggio di richiesta, il messaggio di risposta raggiunge il sito web dei consumatori in due fasi. Nel primo passaggio, il Server OpenID utilizza il metodo redirect HTTP dal Server OpenID per il browser web. Quindi nel passaggio successivo, il browser invia la richiesta HTTP GET il sito web dei consumatori. Ciò è dimostrato in figura :

Esempio di un messaggio checkid_setup response:

```
GET /finish_auth.php?nonce=nC5sKquX&openid.assoc_handle={HMACSHA1}{
46071e25}{Tt8MwQ==}&openid.identity=http://idp.conformix.com/?user=
openidbook&openid.mode=id_res&openid.return_to=http://consumer.conformix.
com:80/finish_auth.php?nonce=nC5sKquX&openid.sig=nXWc+07GLaSf+RghmGubGPPg
lZc=&openid.signed=mode,identity,return_to,sreg.email&openid.sreg.email=r
r@conformix.com HTTP/1.1
```

3.6.5 The check_authentication Request Message

I messaggi di richiesta e risposta di check_authentication sono uno strumento necessario per verificare l'affermazione ricevuta dall'User Agent (browser web). Questo è necessario per garantire che un utente malintenzionato non invia messaggi affermandosi al posto del server OpenID. Si noti quanto segue per i messaggi di check_authentication:

1. Questo messaggio non verrà inviato se esiste già un'associazione tra il sito web dei consumatori e il server OpenID. L'associazione viene inizialmente stabilita mediante “associate message”.
2. Se viene utilizzata un'associazione esistente, il consumer invierà il parametro “openid.assoc_handle ” nella richiesta e il Server OpenID invierà lo stesso handle nella risposta. In tal caso, il sito web dei consumatori saprebbe che il server OpenID ha accettato di utilizzare l’handle esistente.
3. Se il Server OpenID non accetta di utilizzare l’handle dell’associazione fornito dal sito web dei consumatori, la risposta includerà il parametro “openid.invalidate_handle ” il messaggio di risposta e una diverso “openid.assoc_handle ”. Inoltre il consumer utilizzerà il messaggio “check_authentication” per convalidare l'affermazione.
4. Quando si utilizza la modalità dumb, questo messaggio di risposta verrà utilizzato sempre perché i consumer è stateless e nessun record di qualsiasi precedente handle di associazione esiste.

Si noti che questo messaggio è una comunicazione diretta tra il consumer e il server di OpenID e il metodo POST HTTP viene utilizzato per questa comunicazione. La richiesta ha il parametro “openid.mode ” con un valore “check_authentication” e tutti gli altri parametri che facevano parte del messaggio “associate”. Un tipico messaggio avrà l'aspetto seguente:

Esempio:

```
openid.assoc_handle={HMACSHA1}{
460730e1}{zr1gKg==}&openid.identity=http://idp.conformix.com/?user=
openidbook&openid.invalidate_handle={HMACSHA1}{
46071e25}{Tt8MwQ==}&openid.mode=check_authentication&openid.return_
to=http://consumer.conformix.com:80/finish_auth.php?nonce=mAotRbGM&openid
.sig=4hwwyWbPtSAmP2dYxEC+dq605Os=&openid.signed=mode,identity,return_to,s
reg.email&openid.sreg.email=rr@conformix.com
```

3.6.6 The check_authentication Response Message

In risposta al messaggio “check_authentication”, il Server OpenID invierà un messaggio breve con i seguenti parametri:

- **openid.ns**
- **is_valid:** parametro che ha un valore “true ” o “false ”
- **invalidate_handle:** parametro facoltativo e in caso che il parametro “is_valid” è true, il consumer rimuoverà l'handle dal elenco delle handle salvate.

Un tipico messaggio è il seguente:

```
invalidate_handle:{HMAC-SHA1}{46071e25}{Tt8MwQ==}
is_valid:true
```

Da notare che questo messaggio è molto breve e la risposta include solo l'esito positivo o negativo del controllo di autenticazione.

3.7 How OpenID Works: Some Scenarios

Come sapiamo ormai, OpenID viene utilizzato per autenticare un utente su diversi siti web utilizzando le credenziali archiviate nel server di identità scelto. Esistono diversi scenari di come si svolgerà l'autenticazione OpenID.

3.7.1 Scenario One: First Time Login to a Web Site Using OpenID in Dumb Mode

Quando ci loggiamo su un sito web che supporta OpenID per la prima volta succedono un paio di cose. In seguito vediamo una lista di steps (molto in generale, senza entrare nei dettagli):

1. Inseriamo il nostro OpenID URL nella pagina del consumer cliccando il pulsante Login.
2. Il sito web del consumer identifica la locazione del OpenID server, e può usare Yadis per scoprire i servizi che l'URL offre. Il consumer reindirizza il browser (lo user agent) al server OpenID per ottenere le credenziali.
3. Poiché questa è la prima volta che l'utente va presso questo sito web, il server OpenID non sa se questo sito web è attendibile o no. Così il server OpenID visualizzera una schermata di login (login al server OpenID).
4. L'utente indica questo sito web (del consumer) come sito fidato al server OpenID.
5. Il server OpenID reindirizza il browser nella pagina del consumer.
6. Il consumer controlla l'autenticazione direttamente con il server OpenID e l'autenticazione è finita.

3.7.2 Scenario Two: Login to a Trusted Web Site Using OpenID in Smart Mode

L'utente può visitare regolarmente dei siti web. Se questi siti supportano OpenID, si possono segnare come siti “fidati” preso l'Identity Provider. Ecco i step in questo caso:

1. Inseriamo il nostro OpenID URL nella pagina del consumer cliccando il pulsante Login.
2. Il consumer stabilisce un associazione con l'OpenID server, se una tale associazione non esiste.
3. Il sito web localizza l'OpenID server e reindirizza il browser presso il server per ottenere le credenziali. In questo passo può utilizzare anche Yadis.
4. Il server OpenID sa che questo è un consumer fidato e sa quali parametri passargli. Se l'utente è già autenticato presso il server OpenID, il server invierà al consumer le credenziali richieste. L'utente verrà loggato presso il consumer senza ulteriori passaggi.

3.8 Problems Solved by OpenID

OpenID risolve una serie di questioni riguardante l'Identity Management. Alcuni di questi sono i seguenti:

- gli utenti hanno il controllo su quali dati devono essere condivisi con il sito web del consumer.
- OpenID consente, l'utilizzo delle credenziali attraverso tutti i siti web abilitati per OpenID. Così non è necessario creare singolarmente username e password per ogni sito web.
- Ferma i replay attacks utilizzando la variabile nonce una sola volta. Un consumer può ignorare un'affermazione positiva, esaminando il timestamp nella variabile nonce. Se il timestamp è troppo lontano dal tempo corrente, il consumer può respingerla.

4 Altro Scenario

4.1 Attori e scenario:

- **Alice**: *End User*
- **Operfox Explorer**: *User-Agent*
- **Bob**: *Consumer*
- **Carol**: *OpenID Server*
- **http://carol.example.com/Alice**: *Identity*
- **Ive**: *malicious attacker*

Scenario:

- Alice vuole autenticarsi preso Bob
- Bob supporta l'utilizzo del protocollo Open ID
- Alice è autenticata preso l'OpenID Provider Carol, è possiede un account
- http://carol.example.com/Alice, è l'identifier di Alice presso Carol
- Ive vuole spacciarsi per Alice presso Bob

Domande:

Cosa succedere quando Alice inserisce il suo identifier ?

Come fa Bob ad essere sicuro che si tratta veramente di Alice e non di Ive ?

Cosa deve fare Carol per assicurare Bob che si tratta di Alice?

4.1 Act 1:

Alice inserisce la sua identità OpenID, nella form che Bob gli presenta e clicca sul pulsante "Authenticate". Operfox Explorer, lo User Agent di Alice processa la form e la invia ad un CGIs implementato presso Bob (il consumer). Il CGI di Bob normalizza l'url inserito da Alice ed analizza il documento che li viene restituito da tale Url. Fin qui niente di speciale, semplicemente una richiesta di un file via HTTP GET. Quando il consumer (Bob), prende tale file dall'identity url inserito da Alice, lo esplora cercando un tag specifico nella sessione header di tale documento:

"<link rel="opened.server" href=http://carol.example.com/openid-server.cgi">"

Nella modalità "dumb" Bob reindirizza (redirect) l'End User di Alice a tale URL (Carol's OpenID server - http://carol.example.com/openid-server.cgi), aggiungendo alcune informazioni (HTTP GET):

- **openid.mode** = checkid_setup. É una delle modalità possibili. Questo significa che il consumer Bob intende controllare (effettuare un check) un'identità, quella di Alice e sta passando il controllo dello

User Agent al server Carol per effettuare tale controllo. Bob si aspetta che il server risponde, una volta effettuato il controllo.

- **openid.identity** = http://carol.example.com/Alice Questa è l'identifier inserito da Alice, ed è questo che Bob vuole verificare con l'aiuto di Carol.

- **openid.return_to** = http://bob.example.com/comment.cgi?session_id=Alice&nonce=123456

Questo è l'url dove Bob vuole che lo User Agent di Alice si posizioni dopo che Carol ha effettuato l'autenticazione. Inoltre Bob si aspetta dei dati aggiuntivi su questo Url.

4.2 Act 2 Ive:

Vediamo il parametro **openid.return_to.** Questo è il punto dove Bob si aspetta che Alice si posizioni dopo l'autenticazione da parte di Carol. Bob ha da quache parte la session_id di Alice legato a ciò che Alice doveva fare preso Bob. Ma a noi interessa qualcosa in più che una semplice sessione di autenticazione. Ive, per rubare l'identità di Alice deve solo mettersi in ascolto della conversazione Alice – Bob, memorizzare tale conversazione ed effettuare un "replay attack".

Anche se la conversazione fosse crittografata e Ive non potesse vedere cosa ci sia all'interno, per lui sarà ancora possibile effettuare lo "spoofing" dell'identità di Alice. Dobbiamo procurarsi qualcosa in più per assicurarsi che Ive non può farlo, e questo è il parametro extra "**nonce**".

Il consumer Bob, inserisce un identificatore, un numero random, all'Url per ogni richiesta di autenticazion che fa. Grazie a questo valore, "nonce", ogni richiesta di autenticazione ha il suo valore univoco e quindi per Ive sarà impossibile effettuare un "replay attack" poiché il valore return_to è univoco a quella sessione.

4.3 Act 3 Torniamo da Carol

Torniamo da Carol che ha del lavoro da svolgere. Bob le ha chiesto di confermare che Alice veramente possiede l'url che ha inserito, reindirizzando lo user agent di Alice presso Carol. Carol, adesso, ha il controllo dello User Agent di Alice e quindi può autenticare che ci sia veramente Alice dall'altra parte dello schermo.

Come fa ???

Come dicevamo in precedenza questo non fa parte del protocollo OpenID. Per OpenID questa verifica è una scatola nera. OpenID di occupa dell'identità di Alice e non della fiducia che Carol ha in Alice. Quello che ci interessa, ed è quello che interessa anche a Bob, è quello che ci risponde Carol riguardante Alice. Bob si fida di Carol, altrimenti che senso avrebbe interrogare Carol per l'identità di Alice. Ciò che sappiamo è che Alice è la stessa Alice che Carol dice di lei. Oltre a questo, Alice non può far finta di essere qualcun'altra, almeno non senza l'aiuto di Carol. E se cosi fosse dobbiamo smettere di prestare attenzione a Carol ed a tutti gli utenti che usano Carol come OpenId Provider.

Carol si decide che sia davvero Alice dall'altra estremità dello User Agent. Ora, Carol deve solo convincere Bob che sia davvero Alice in linea, e anche convincerlo che essa è davvero Carol. Facile, vero? Il primo step in questo processo è per Carol quello di reindirizzare lo User Agent di Alice nel cgi di Bob http://bob.example.com/comment.cgi?session_id=Alice&nonce=123456 (return_to url indicato da Bob) con in aggiunta questi parametri (GET):

- **openid.mode** = id_res Questo valore indica che Carol afferma che Alice possiede l'identifier da lei dichiarato. Questo valore può essere anche "cancel" che sta ad indicare che Alice non vuole continuare l'autenticazione.

- openid.return_to = http://bob.example.com/comment.cgi?session_id=Alice&nonce=123456
Lo stesso URL che Bob ha inviato a Carol nella richiesta di autenticazione per Alice.

- **openid.identity** = http://carol.example.com/Alice L'identifier che Alice dice di possedere e che Carol ha affermato.

- **openid.signed** = mode,identity,return_to. Questo è l'elenco dei parametri a cui dobbiamo fornire una firma. Ovviamente vogliamo firmare la modalità e l'identità che stiamo autenticazione, ma anche l'URL di return_to per impedire attacchi di tipo replay, come discusso prima.

- **openid.assoc_handle** = *opaque handle*

- **openid.sig** = *base 64 encoded HMAC signature*. (HMAC = **keyed-hash message authentication code**)

Vediamo in dettaglio gli ultimi due parametri:

Il primo **assoc_handle** è definito nel protocollo OpenID come "an opaque handle". Questa è una handle per il segreto. Carol deve poter prendere questo handle opaco e ricercare, internamente, quali segreti ha utilizzato per la firma dell'affermazione per Bob. Come lei fa questo è un problema interno di Carol, ma l'unica cosa che lei ha bisogno e quello di distinguere tra un normale assoc_handles e quella della modalità stateless.

Carol costruisce questo segreto e lo lega al assoc_handle, ma a che serve questo segreto? Bene, viene utilizzato per creare il secondo elemento di interesse, la firma. Carol raggruppa insieme i campi che abbiamo detto in precedenza e che intende firmare (nel nostro caso: mode, identity e return_to) e quindi esegue l'algoritmo HMAC-SHA1* per la firma su di esso, utilizzando il segreto che ha legato alla assoc_handle come chiave di crittografia per HMAC. Questo genera una firma che ella quindi codifica in base 64 da inviare come parametro a Bob.

***HMAC** (**keyed-hash message authentication code**) è una tipologia di codice per l'autenticazione di messaggi basata su funzione di hash, utilizzata in diverse applicazioni legate alla sicurezza informatica. Tramite HMAC è infatti possibile garantire sia l'integrità che l'autenticità di un messaggio. HMAC utilizza infatti una combinazione del messaggio originale e una chiave segreta per la generazione del codice. Peculiare di HMAC è il non essere legata a nessuna funzione di hash particolare, questo per rendere possibile una sostituzione della funzione nel caso fosse scoperta debole. Nonostante ciò le funzioni più utilizzate sono MD5 e SHA-1

4.4 Showtime Bob! Or is it?

Grande! Bob ha ora un'affermazione firmata che Alice è davvero chi lei afferma di essere e tutto ciò è andato bene, giusto? Bene ci sono ancora dei dubbi. In primo luogo, Bob effettivamente non può controllare la firma da solo, dopo tutto non consce il segreto. Tutto quello che possiede è questa handle che con essa egli non può fare nulla e una firma che può o non può essere valida. In secondo luogo, tutta questa informazione proviene direttamente dallo User Agent di Alice in ogni caso, e l'intero set di bit possa essere compromesso. Bob a questo punto non sa se questa informazione ha da fare con Carol o no. È quindi?

Come abbiamo detto prima, Bob lavora in modalità dumb e quindi deve ricontattare Carol. Il CGI che lo User Agent di Alice ha contattato presso Bob aprirà una sessione HTTP POST a Carol un'altra volta prima di fidarsi dell'autenticazione di Alice. Questa ultima sessione avviene direttamente fra Bob e Carol e lo User Agent di Alice non ha a che fare. Bob invia via il metodo HTTP POST, i seguenti parametri al cgi di Carol (http://carol.example.com/openid-server.cgi):

- **openid.mode** = check_authentication: dicendo a Carol che intende confermare quanto

Carol ha detto di Alice.

- **openid.signed** = mode,identity,return_to: firmati con la firma di Bob

- **openid.assoc_handle** = *opaque handle*: Lo stesso di prima

- **openid.sig** = *base 64 encoded HMAC signature*

Quando Carol riceve questa richiesta andrà a fare tutto il lavoro che ha già fatto in precedenza.

Lei aggiungerà all'elenco dei parametri firmati insieme ancora una volta, ricercherà il segreto che ha legato alla assoc_handle fornito e creare una firma HMAC-SHA1 dei parametri utilizzando questo segreto come la chiave di crittografia, proprio come ha fatto prima. Quindi lei confronterà la firma digitale con la firma che Bob ha fornito nel suo messaggio. Se essi corrispondono, allora Carol sa che deve essere stata lei che ha inviato l'affermazione originale (o altrimenti, qualcuno che conosce i suoi segreti), e restituirà un file di testo normale da Bob con la sua risposta definitiva: "**is_valid:true**".

Ovviamente se le due firme non corrispondono Carol invierà il messaggio "**is_valid:false**", e Bob saprà che qualcosa è andato storto.

4.5 Bob e Javascript - (AJAX)

In questo caso viene utilizzato il messaggio "checkid_immediate" il quale come "checkid_setup" openid.mode è inviato al server OpenID di Carol con tutta l'informazione necessaria per autenticare l'identifier di Alice. L'unica differenza di questa modalità è che, come indicato dal nome, viene effettuata immediatamente senza prendere il controllo dello User Agent di Alice. Se il server può autenticare in loco, tutto continua come normale e il consumatore (Bob) deve seguire con un messaggio "check_authentication".

Tuttavia, se il server OpenID di Carol non può fare un'affermazione positiva sull'identità, "checkid_immediate" restituisce quindi istantaneamente un'asserzione fallita e un parametro singolo "openid.user_setup_url" invece di assumere il controllo dello User Agent. A questo punto, il consumatore (Bob) può decidere che cosa fare. Può reindirizzare lo User Agent nell'Url "openid.user_setup_url" fornito, o fare ciò con un popup. Il punto chiave, tuttavia, è che il consumatore ha il controllo di come dovrebbe comportarsi lo User Agent.

Quando un utente tenta di autenticarsi, invece di inviare la form direttamente al server OpenID server tramite lo User Agent, si usa JavaScript per aprire un HTTP Request al server OpenID. E anziché inviare una modalità "checkid_setup", che si aspetta di poter assumere il controllo dello User Agent, inviano invece un messaggio "checkid_immediate". In questo modo si può aprire una nuova finestra web (popup) con l'url "**openid.user_setup_url**" dove l'utente può finalizzare i passi necessari al server OpenID per consentire l'autenticazione. Seguento, con un altro HTTP Request per inviare il comando "check_authentication" l'intero processo ha avuto luogo senza interrompere lo User Agent minimamente.

4.6 Associate – Smart mode

La procedura come vista finora richiede un utilizzo massiccio della rete. Inoltre anche le altre risorse del server come la CPU e la Memoria sono sovraccaricate dal lavoro che devono fare, poiché per ogni richiesta di autenticazione devono generare una chiave segreta.

Se il consumer (Bob) potrebbe ricordare anche per un po di che cosa stava parlando, egli può risparmiare a Carol un sacco di lavoro ed ad entrambi molta banda. L'idea base è semplice: in modalità

“dumb", il consumer (Bob) semplicemente riporta indietro la handle (che non tiene traccia dello stato - stateless) che gli è stato inviato da Carol . Ma cosa succederebbe se Bob potrebbe effettivamente ricordasse il segreto condiviso con Carol utilizzandolo e riutilizzandolo questo segreto condiviso per un lasso di tempo ogni volta che ha bisogno di parlare con Carol?

Questa è la modalità “**Smart**”

Il consumer (Bob) stabilisce un segreto condiviso con l’OpenID server di Carol in anticipo sui tempi e ricordandolo questo segreto per qualche periodo di tempo ragionevole. Egli stabilisce il segreto condiviso inviando una richiesta POST al server di Carol,

http://carol.example.com/openid-server.cgi con questi parametri:

- **openid.mode** = associate Questo indica a Carol che vuole condividere una chiave segreta con lei.

- **openid.assoc_type** = HMAC-SHA1

+ The type of secret he wants to share; only HMAC-SHA1 is currently supported.

- openid.session_type = *blank* Indica come il segreto deve essere stabilito, un valore vuoto significa in chiaro il che non è 100% sicuro. "DH-SHA1" indica che verrà usato il protocollo Diffie – Hellman per lo scambio delle chiavi

Quando Bob effettua questo scambio? Può essere fatto la prima volta che qualcuno chiede di autenticare il suo identifier con Carol, o se tale chiave segreta è scaduta. Ma la cosa importante da notare qui è che, questo processo è complettamente indipendente ad ogni richiesta di autenticazione fatta, ache se può essere attivata da una tale richiesta.

Una volta che Bob ha inviato il suo messaggio di “associate request”, Carol genererà una chiave segreta e condivisa (come avrebbe fatto anche prima) e lo aggiunge a “assoc_handle”. Diversamente dalla modalità “dumb” in questa modalita questo handle è un handle stateful, ovvero tiene traccia dello stato della conversazione.Carol quindi risponde a Bob con un documento * chiave = valore * formattato in risposta alla sua richiesta POST che contiene tutti i normali parametri vari. I due parametri magici, tuttavia, sono il * assoc_handle * e * mac_key * (o * enc_mac_key * se è stato utilizzato Diffie-Hellman). Con questi due elementi, Bob ora può tenere traccia, internamente, a tale handle e la chiave segreta condivisa.

Ma a che servono questi due valori ? Ora, ogni volta che Bob ha bisogno di autenticare l'identità con Carol, egli invierà la handle con il messaggio "checkid_setup" o "checkid_immediate" invece di aspetta da Carol la generazione della handle ogni volta. Questo consente a Carol di risparmiare del lavoro. Una volta Bob ottiene una risposta firmata da Carol (supponendo un'affermazione positiva), egli non ha più bisogno di inviare una richiesta "check_authentication" perché sa gia la chiave segreta condiviso che è stata utilizzata per firmare l'affermazione e lo può controllare da solo. Prima Bob doveva verificare con Carol perché non aveva nessuno modo di sapere se la firma utilizzata è era autentica (di Carol) oppure falsa e quindi doveva interagire con Carol per tale controllo. Ma ora poiché Bob sa già la chiave segreta utilizzata per la firma può controllare lui da solo la validità della firma.

5 Flusso del protocollo