UNIVERSITÀ DEGLI STUDI DI MILANO

DIPARTIMENTO DI INFORMATICA – SEDE DI CREMA

VIA BRAMANTE 65 – 26013 CREMA (CR) – ITALIA

OPENID CONNECT 1.0

VERIFICA FORMALE DEL PROTOCOLLO IN HLPSL

Docente: Prof. Ernesto DAMIANI

Studente: Vincenzo G. CALABRÒ

Matr: 808052

Indice

Introduzione

Lo sviluppo dei servizi web, negli ultimi anni, si è concentrato sull'integrazione e il riutilizzo delle risorse e dei servizi distribuiti in rete. Questa impostazione progettuale è testimoniata dall'avvento delle *Service Oriented Architecture (SOA)*, adatte a garantire l'interoperabilità tra molteplici sistemi software, scritti in diversi linguaggi di programmazione, ed implementati su differenti piattaforme hardware. I servizi, resi disponibili da tali sistemi, possono essere facilmente combinati e integrati tra di loro per formarne altri più complessi.

Contestualmente alla diffusione di sistemi software distribuiti, è maturata l'esigenza di disporre di meccanismi di autenticazione unica (*Single Sign-On, SSO*) per il controllo dell'accesso a più risorse indipendenti, ma unificate sotto uno stesso dominio applicativo.

Il Single Sign-On, mediante un server di autenticazione centralizzato, semplifica notevolmente la gestione degli accessi ai diversi servizi e permette, allo sviluppatore ed all'utente finale, di risparmiare tempo e di non doversi più preoccupare della gestione di numerose combinazioni e policy di credenziali (username e password).

In questo scenario il servizio di autenticazione centralizzato diventa particolarmente critico dal punto di vista dell'affidabilità, p.e. la mancata disponibilità del servizio impedirebbe l'accesso a tutte le risorse appartenenti ad un dato dominio, e della sicurezza, in quanto la conseguenza negativa derivante dalla compromissione delle credenziali dell'utente verrebbe amplificato, in quanto il SSO consente l'accesso a molteplici risorse.

Di conseguenza, i sistemi di Single Sign-On sono soggetti ad attacchi volti principalmente all'accesso illegittimo di risorse e servizi, ovvero all'impedimento della fruizione.

È ben noto, tuttavia, che la progettazione di protocolli di autenticazione affidabili e sicuri, a dispetto della loro apparente semplicità, è un'attività complessa e soggetta ad errori. Diversi protocolli di Single Sign-On si sono rivelati vulnerabili anche dopo anni dalla loro implementazione e

pubblicazione (p.e. Kerberos versione 3[1] e Kerberos versione 4[2], e più recentemente la prima implementazione del protocollo di autenticazione di Google, basato su SAML SSO[3]).

Talvolta gli attacchi sono resi possibili da dettagli trascurati in fase di progetto. Allo scopo di ridurre al minimo le vulnerabilità dei sistemi di autenticazione, si sta diffondendo l'uso di tecniche di verifica formale della specifica astratta dei protocolli. La comunità internazionale di ricerca nel campo dei metodi formali ha recentemente proposto diversi strumenti automatici a supporto di tale approccio. Tra i tool più maturi AVISPA[4] è uno strumento integrato per la verifica automatica di proprietà di sicurezza di protocolli e applicazioni. Il tool fornisce il linguaggio HLPSL (High Level Protocol Specification Language) per la specifica dei protocolli ed integra diversi moduli back-end di verifica che implementano varie tecniche di analisi formale.

Lo scopo di questo lavoro è l'analisi formale con AVISPA di OpenID Connect 1.0, uno dei più recenti protocolli di Single Sign-On rilasciato nella versione standard il 24 febbraio 2014.

OpenID Connect 1.0 è un semplice "identity layer", basato sul protocollo OAuth 2.0, che permette ai Client di verificare l'identità dell'End-User attraverso l'autenticazione eseguita da un Authorization Server, nonché di ottenere informazioni di base dell'End-User stesso.

OpenID Connect consente a qualsiasi tipo di Client, compresi i client Web-based, mobile e JavaScript, di richiedere e ricevere informazioni sulle sessioni autenticate e gli utenti finali. La specifica della suite è estensibile, permettendo ai partecipanti di utilizzare le funzioni opzionali come la crittografia dei dati d'identità, la scoperta dei fornitori di OpenID e la gestione della sessione. Essendo un sistema SSO recente è possibile che presenti delle vulnerabilità ancora sconosciute.

Il lavoro è strutturato come segue. Nel Capitolo 1 verrà presentato AVISPA: dopo una breve descrizione comprensiva d'esempi del linguaggio HLPSL,

[1] S. M. Bellovin and M. Merritt. Limitations of the Kerberos Authentication System. Computer Communication Review, 20:119–132, 1990.
[2] S. Hartman T. Yu and K. Raeburn. The Perils of Unauthenticated Encryption: Kerberos Version 4. In NDSS. The Internet Society, 2004.
[3] A. Armando, R. Carbone, L. Compagna, J. Cuéllar, and M. Llanos Tobarra. Formal Analysis of SAML 2.0 Web Browser Single Sign-On: Breaking the SAML-based Single Sign-On for Google Apps. In Vitaly Shmatikov, editor, FMSE, pages 1–10. ACM, 2008.
[4] L. Viganò. Automated Security Protocol Analysis with the AVISPA Tool. Electr. Notes Theor. Comput. Sci., 155:61–86, 2006.

saranno illustrate le proprietà dei back-end attualmente integrati. Il Capitolo 2 offrirà una presentazione informale di OpenID Connect. Il lavoro di analisi del protocollo sarà illustrato in dettaglio nel Capitolo 3. In primo luogo verrà introdotta una formalizzazione in notazione Alice-Bob del protocollo di OpenID Connect. Successivamente sarà fornita la specifica HLPSL corrispondente, sulla quale verrà condotta l'analisi automatica di diverse proprietà di sicurezza. Il protocollo nella sua formulazione base, come si vedrà, è vulnerabile all'intercettazione delle informazioni di autenticazione di sessione. Pertanto, alla luce delle considerazioni sulla sicurezza, verrà proposta una soluzione arricchita per rimuovere il bug rilevato. Infine, nel ultimo Capitolo saranno esposte le conclusioni ed alcune considerazioni in merito ai possibili sviluppi futuri del lavoro presentato.

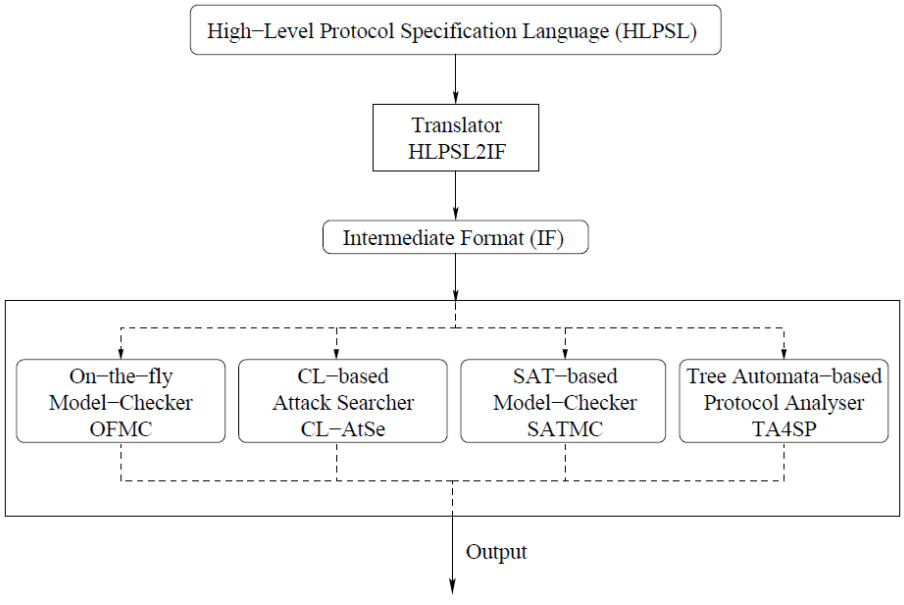
1 AVISPA

AVISPA[5] (Automated Validation of Internet Security Protocols and Applications) è un tool integrato per la validazione automatica di protocolli ed applicazioni di sicurezza per sistemi distribuiti.

È stato sviluppato nell'ambito di un progetto avente l'obiettivo dello sviluppo di una tecnologia che permetta di accelerare l'elaborazione di protocolli di rete sicuri e la diffusione di applicazioni distribuite avanzate basate su di essi. AVISPA è già stato utilizzato con successo per l'analisi di diversi protocolli (IKE, ad esempio) nel corso della standardizzazione da parte di enti quali IETF, ITU e W3C. Risultati sperimentali, ottenuti dall'analisi di una vasta libreria di protocolli di sicurezza per Internet, indicano inoltre che AVISPA rappresenta lo stato dell'arte per l'analisi dei protocolli di rete.

L'applicazione fornisce un linguaggio di alto livello (High Level Protocol Specification Language, o HLPSL) per la specifica formale dei protocolli e delle loro proprietà di sicurezza. Inoltre, AVISPA integra più componenti che implementano molteplici tecniche di analisi automatica.

La struttura dell'applicazione è illustrata di seguito



Architettura di AVISPA

[5] The AVISPA Team. AVISPA v1.1 User Manual, 2006. http://www. avispa-project.org/publications.html.

In particolare AVISPA è costituita da diversi moduli: un traduttore chiamato HLPSL2IF per trasformare le specifiche HLPSL scritte dagli utenti in specifiche IF (Intermediate Format) e quattro differenti back-end (OFMC, CL-AtSe, SATMC, TA4SP) in grado di analizzare le specifiche IF per la verifica dei protocolli di rete.

1.1 Il linguaggio HLPSL

HLPSL è un linguaggio espressivo utile per la modellazione della comunicazione e dei protocolli di sicurezza. Esso trae le sue origini dalla Logica Temporale delle Azioni (Temporal Logic of Actions, o TLA) di Leslie Lamport [6]. TLA è un linguaggio elegante e potente molto utile per specificare sistemi concorrenti. Informazioni più dettagliate sull'argomento sono disponibili sul sito web *http://www.avispa-project.org*.

Numerosi linguaggi per la specifica di protocolli sono basati sulla semplice notazione Alice-Bob. Tale notazione è molto intuitiva e compatta ed è utile ad illustrare lo scambio di messaggi che avviene nel corso di una normale esecuzione di un dato protocollo. Tuttavia, spesso risulta troppo informale e non sufficiente per catturare la sequenza di eventi che dev'essere considerata nel caso dei protocolli di rete. È questo il motivo per cui AVISPA ha bisogno di un linguaggio più espressivo come HLPSL.

1.1.1 Caratteristiche Fondamentali

La descrizione completa della sintassi di HLPSL si trova in [7]. Di seguito sono riportati i soli aspetti essenziali del linguaggio.

Nonostante HLPSL risulti più efficace rispetto alla semplice notazione Alice-Bob, scrivere le specifiche dei protocolli in HLPSL non è altrettanto semplice. HLPSL è un linguaggio basato sui ruoli. I protocolli scritti in HLPSL, infatti, sono definiti ruolo dopo ruolo piuttosto che messaggio dopo messaggio come invece avviene utilizzando la notazione Alice-Bob. In questo modo le specifiche in HLPSL appaiono molto meno ambigue, ma più difficili da leggere. Allo stesso modo, alle volte risulta difficile, per i progettisti, verificare se la specifica HLPSL che hanno scritto corrisponde al protocollo in notazione Alice-Bob che hanno in mente.

---

[6] L. Lamport. The Temporal Logic of Actions. ACM Trans. Program. Lang. Syst., 16(3):872–923, 1994.

[7] The AVISPA Team. Deliverable D2.1: The High Level Protocol Specification Language, 2003. http://www.avispa-project.org/publications.html.

HLPSL è un linguaggio basato sulla logica temporale. I protocolli, pertanto, sono modellati come sistemi di transizioni mediante la descrizione dello stato del sistema e la specifica delle modalità con cui tale stato può cambiare. In HLPSL, come già accennato, la specifica del sistema è suddivisa in ruoli. Per ogni ruolo viene definito l'insieme delle sue variabili tra le quali vi sono le variabili di stato. Lo stato di un ruolo è determinato, pertanto, dal valore di tutte le sue variabili di stato. A partire da tale definizione, è possibile specificare i concetti di predicato di stato e di predicato di transizione. Un predicato di stato (*state predicate*) è una formula della logica del prim'ordine sulle costanti e sulle variabili di stato di un ruolo. Un esempio è **`State = 0`**. I predicati di stato modellano lo stato di un dato ruolo in un certo istante di tempo. I predicati di transizione, al contrario, modellano l'evoluzione dello stato di un ruolo nel momento in cui avviene una transizione da uno stato ad un altro. Un predicato di transizione (*transition predicate*) è simile ad un predicato di stato ma può contenere *primed variables*. Le *primed variables* servono per riferirsi alle variabili nello stato successivo a quello attuale: data una variabile **X**, con **X** ci si riferisce al suo valore nello stato corrente, con **`X'`** al suo valore nel prossimo stato. Un esempio di predicato di transizione è **`X':= 5`** che rappresenta l'assegnamento del valore **5** alla variabile **X**.

HLPSL è un linguaggio *tipato*: ogni variabile o costante può avere un unico tipo. Di seguito sono riportati i tipi di base principali disponibili in HLPSL (i tipi di dato `channel` e `protocol_id` verranno illustrati in seguito):

- `agent`: tipo di dato per gli agenti
- `nat`: tipo di dato per i numeri naturali
- `text`: tipo di dato per stringhe di bit non interpretate (generalmente è il tipo utilizzato per i nonces)
- `public_key`: tipo di dato per chiavi pubbliche (data una chiave pubblica k, è possibile ottenere la chiave privata corrispondente mediante `inv(k)`)
- `symmetric_key`: tipo di dato per chiavi simmetriche
- `hash_func`: tipo di dato per funzioni unidirezionali

Utilizzando tali tipi di base e le operazioni di concatenazione e cifratura è possibile costruire tipi aggregati. Ad esempio `{agent.text}_symmetric key` è un tipo di dato aggregato costituito da due tipi di base concatenati mediante l'operatore '.' e, successivamente, cifrati mediante una chiave simmetrica.

In HLPSL, infine, la *comunicazione* è *sincrona*. Gli agenti comunicano tra loro mediante lo scambio di messaggi attraverso i canali di comunicazione. I

messaggi possono essere costituiti dai tipi di base oppure dai tipi aggregati appena descritti.

I canali sono variabili attraverso le quali avviene la comunicazione tra gli agenti. Il tipo di dato `channel` accetta un attributo che specifica il modello di canale che si vuole utilizzare. Attualmente, l'unico modello supportato è il modello Dolev-Yao, in cui l'attaccante ha il pieno controllo della rete: può intercettare, analizzare, modificare e inviare qualsiasi messaggio a qualunque agente egli voglia impersonandone qualsiasi altro. Nel modello Dolev-Yao l'attaccante è così potente che non risulta necessario differenziarlo dalla rete: l'attaccante è la rete. In questo modo ogni messaggio inviato da un agente onesto viene ricevuto dall'attaccante ed ogni messaggio ricevuto da un agente onesto proviene dall'attaccante.

1.1.2 Ruoli

In HLPSL le specifiche dei protocolli sono organizzate e suddivise in ruoli. Un ruolo può essere considerato come la descrizione di un comportamento. Alcuni ruoli (ruoli base, *basic roles*) rappresentano ogni agente partecipante allo svolgimento di un protocollo e ne descrivono le azioni. Altri (ruoli composti, *composed roles*) istanziano i ruoli base al fine di modellare l'interazione degli agenti nel corso dell'intera esecuzione del protocollo (ruoli sessione, *session roles*) oppure definiscono le sessioni del protocollo di effettivo interesse (ruoli ambiente, *environment roles*).

La definizione di un ruolo, infine, generalmente consiste nei seguenti elementi:

- dichiarazione del nome del ruolo, della lista dei suoi parametri e, nel caso dei ruoli base, dell'agente che il ruolo rappresenta
- dichiarazione delle variabili locali
- inizializzazione delle variabili locali
- dichiarazione delle conoscenze dell'attaccante (nel caso del ruolo ambiente)
- opzionalmente, una *transition section* (per i ruoli base) o una *composition section* (per i ruoli composti)

1.1.2.1 Ruoli Base

I protocolli, come già accennato, sono modellati come sistemi di transizioni mediante le nozioni di stato, transizione ed i rispettivi predicati definiti nella sezione precedente. Per un dato ruolo base, i valori delle sue variabili costituiscono il suo stato e la transition section descrive le transizioni valide.

Attualmente le uniche transizioni supportate sono le transizioni a reazione immediata le quali sono della forma X =|> Y, dove X è un evento mentre Y è un'azione. Queste modellano il fatto che, nel momento in cui un certo evento X risulta verificato, immediatamente (o meglio, simultaneamente), l'azione Y viene eseguita.

Mediante le transizioni a reazione immediata è possibile modellare la sincronicità della comunicazione in HLPSL: per come sono definite tali transizioni, infatti, le azioni di ricezione (a sinistra del simbolo =|>) avvengono simultaneamente alle corrispondenti azioni di invio (a destra del simbolo =|>).

La comunicazione tra gli agenti avviene per mezzo delle variabili di tipo channel. Per convenzione, ai canali vengono generalmente assegnati nomi come SND e RCV e, di conseguenza, l'invio e la ricezione di un messaggio Msg sono descritti rispettivamente da SND(Msg) e RCV(Msg). Nei modelli di canale Dolev-Yao, inoltre, si usa raggruppare la ricezione di un messaggio e l'invio della risposta corrispondente in un'unica transizione. Tali transizioni assumono generalmente la seguente forma:

```
State = 0 /\ RCV(Request) =|> State':=1 /\
SND(Reply)
```

Infine, le variabili all'interno delle azioni di ricezione possono servire a due propositi: da un lato possono restringere i messaggi accettati (utilizzando variabili già istanziate), dall'altro possono essere istanziate dai messaggi in arrivo (mediante le primed variables). Supponendo, ad esempio, vi sia la seguente transizione all'interno della descrizione di un ruolo:

```
State = 0 /\ RCV({Kab'}_Kas) =|> State':=1
```

quello che si può osservare è che il messaggio che ci si aspetta di ricevere dev'essere cifrato con la chiave Kas: il fatto che questa variabile non sia una primed variable indica che il messaggio deve utilizzare lo stesso valore che la variabile Kas ha attualmente. Il contenuto del messaggio cifrato, invece, può essere arbitrario. Qualunque esso sia verrà assegnato alla variabile Kab che, infatti, è una primed variable.

In conclusione, ogni ruolo di base è indipendente dall'altro, descrive le informazioni che il partecipante può utilizzare inizialmente (parametri), il suo stato iniziale e le differenti modalità secondo le quali lo stato può cambiare (transizioni) mediante la comunicazione con altri ruoli attraverso i canali.

Un esempio di definizione di un ruolo base è il seguente:

```
role alice (A,B,S : agent,
Kas : symmetric_key, SND,RCV : channel(dy) )
played_by A
def=
local
State : nat,
Kab : symmetric_key init
State:=0 transition
...
end role
```

In questo caso, il ruolo alice accetta tre agenti (A, B ed S), una chiave simmetrica (Kas) e due canali (SND e RCV) come parametri. La dichiarazione played by A indica che il ruolo rappresenterà l'agente A. Nella definizione del ruolo sono dichiarate le variabili locali State e Kab e la variabile State è, successivamente, inizializzata a 0. La sezione transition (qui non riportata) conterrà le transizioni valide per i cambiamenti di stato del ruolo.

1.1.2.2 Ruoli Composti

I ruoli composti non hanno una transition section. Al contrario, è presente una composition section all'interno della quale avviene l'inizializzazione di altri ruoli, siano essi ruoli base o ruoli composti a loro volta. Il ruolo composto sessione (session role) serve ad istanziare i ruoli base passando loro come argomenti i parametri che definiscono le loro conoscenze iniziali. Successivamente i ruoli base vengono eseguiti insieme, solitamente in parallelo, al fine di modellare l'esecuzione del protocollo.

Il seguente è un esempio di ruolo sessione:

```
role session (
A,B,S : agent,
Kas,Kbs : symmetric_key )
def=
local
SA,RA,SB,RB,SS,RS : channel(dy)
composition alice(A,B,S,Kas,SA,RA)
/\ bob(B,A,S,Kbs,SB,RB)
/\ server(S,A,B,Kas,Kbs,SS,RS)
end role
```

In questo caso si può osservare come all'interno dei ruoli sessione vengano solitamente dichiarati tutti i canali utilizzati dai ruoli base.

Il ruolo composto di più alto livello è il ruolo ambiente (environment role) che contiene le costanti globali e la composizione di uno o più sessioni di interesse per il modello del protocollo che si vuole descrivere. L'attaccante (a cui ci si riferisce mediante la costante i) può prendere il posto di un utente legittimo in una sessione del protocollo. Un'asserzione che indica le sue conoscenze iniziali: generalmente egli conosce i nomi di tutti gli agenti, tutte le chiavi pubbliche, le sue chiavi private, le chiavi condivise con altri agenti e tutte le funzioni pubblicamente note.

Un esempio di ruolo ambiente è riportato di seguito:

```
role environment()
def=
const
a,b,s : agent, kas,kbs,kis : symmetric_key
intruder_knowledge = {a,b,s,kis}
composition session(a,b,s,kas,kbs)
/\ session(a,i,s,kas,kis)
/\ session(i,b,s,kis,kbs)
end role
```

In questo caso l'attaccante, con il fine di violare il protocollo, gioca il ruolo di un utente legittimo sia nella seconda sessione che nella terza.

Al termine di una specifica HLPSL di un protocollo, infine, vi è sempre un asserzione che serve ad istanziare il ruolo ambiente:

```
environment()
```

1.1.3 Goal di Sicurezza

Dopo aver definito l'insieme di ruoli necessario alla descrizione del protocollo, le specifiche in HLPSL prevedono la dichiarazione dei goal di sicurezza. Essi, in combinazione con i goal facts con cui vengono dettagliate le transizioni dei ruoli base, descrivono le condizioni che indicano un attacco. Attualmente HLPSL prevede solamente goal di autenticazione e segretezza.

Il goal di segretezza secrecy of kab, ad esempio, insieme al goal fact corrispondente secret(Kab, kab, {A,B}), indica che una certa variabile Kab deve rimanere sempre nota soltanto agli agenti A e B. Nel caso in cui un attaccante venisse in possesso di Kab e non risultasse tra gli agenti legittimati a conoscere il valore di tale variabile, ne risulterebbe una violazione di sicurezza. L'argomento kab è di tipo protocol id ed identifica il dato da mantenere segreto (Kab, in questo caso).

I goal di autenticazione previsti da HLPSL sono di due tipi e corrispondono rispettivamente alle definizioni di Gavin Lowe di autenticazione forte e debole. Nel caso dell'autenticazione forte, nessun agente accetta lo stesso valore una seconda volta dallo stesso agente con cui è in comunicazione. Nel caso dell'autenticazione debole, invece, non vi è alcuna protezione contro le ripetizioni. Ad esempio, il goal authentication on alice bob nb, insieme ai goal facts witness(B, A, alice bob nb, Nb) e request(A, B, alice bob nb, Nb), indica che Alice deve autenticare (nel senso dell'autenticazione forte) Bob sul valore del nonce Nb: in particolare, Alice richiede che Bob esista e concordi con lei sul valore di Nb. Il goal weak authentication on bob alice na, invece, insieme ai goal facts witness(A, B, bob alice na, Na) e wrequest(B, A, bob alice na, Na), indica che Bob deve autenticare (nel senso dell'autenticazione debole) Alice sul valore del nonce Na: in questo caso cade la necessità che Alice esista, è sufficiente che sia esistita nel passato e abbia concordato con Bob sul valore di Na. Gli argomenti alice bob nb e bob alice na sono di tipo protocol id e per convenzione sono costituiti dal nome del ruolo autenticatore, il nome del ruolo che dev'essere autenticato e il nome della variabile sul cui valore avviene l'autenticazione, concatenati insieme.

1.2 I Back-End

Attualmente AVISPA integra quattro back-end:

- OFMC (On-the-Fly Model-Checker)
- CL-AtSe (Constraint-Logic-based Attack Searcher)
- SATMC (SAT-based Model-Checker)
- TA4SP (Tree Automata tool based on Automatic Approximations for theAnalysis of Security Protocols)

Nelle sezioni successive saranno brevemente presentate le caratteristiche più importanti di ognuno di essi. Opportuni riferimenti bibliografici saranno messi a disposizione per ulteriori approfondimenti. Il formato di output, inoltre, è lo stesso per tutti i moduli.

1.2.1 OFMC

Il back-end On-the-Fly Model-Checker (OFMC)[8] esplora il sistema di transizioni descritto dalla specifica IF di un protocollo in maniera demand-driven (o on-the-fly, da cui prende il nome).

Il modulo offre, inoltre, la possibilità (mediante l'opzione -p) di esplorare manualmente l'albero delle transizioni per un dato protocollo. Tale opzione è utile, in particolare, per verificare che la specifica permetta agli agenti di effettuare tutti i passi necessari per l'esecuzione di una sessione legittima del protocollo.

OFMC può essere utilizzato non solo per rilevare efficientemente se un protocollo è soggetto ad attacchi, ma anche per provarne la correttezza per un numero limitato di sessioni. Tale vincolo è dovuto al fatto che il problema della verifica della sicurezza di un protocollo è stato dimostrato essere NP-completo [9].

1.2.2 CL-AtSe

Ugualmente il modulo Constraint-Logic-based Attack Searcher (CL-AtSe)[10], come OFMC, può essere utilizzato sia per rilevare se un protocollo è vulnerabile ad un qualche attacco, sia per verificarne la correttezza (per un numero limitato di sessioni). CL-AtSe fornisce la traduzione dalla specifica IF di un qualsiasi protocollo di sicurezza in un insieme di vincoli successivamente utilizzati per l'individuazione di eventuali attacchi. In particolare, ogni passo del protocollo è modellato da vincoli sulle conoscenze dell'attaccante.

Nel corso della lettura di una specifica IF di un protocollo, CL-AtSe attua numerose ottimizzazioni per ridurre (e spesso eliminare) ridondanze o rami inutili al fine di limitare il numero di passi che devono essere controllati. Tale semplificazione è molto importante nel caso di protocolli complessi poiché riduce il tempo richiesto per l'analisi da parte del back-end.

CL-AtSe, infine, è in grado di rilevare attacchi basati sulle proprietà algebriche delle operazioni di esponenziazione modulare e di XOR.

---

[8] S. Modersheim D. A. Basin and L. Viganò. OFMC: A Symbolic Model Checker for Security Protocols. Int. J. Inf. Sec., 4(3):181–208, 2005.
[9] M. Rusinowitch and M. Turuani. Protocol Insecurity with Finite Number of Sessions is NP-complete. In Theoretical Computer Science, pages 174–190, 2001.
[10] M. Turuani. The CL-AtSe Protocol Analyser. In RTA, pages 277–286, 2006.

1.2.3 SATMC

Il back-end SAT-based Model-Checker (SATMC)[11] è stato sviluppato a partire dall'idea che le tecniche di model-checking basate sul problema NP-completo SAT potessero essere sfruttate per individuare eventuali attacchi ai protocolli di rete. SATMC, in particolare, riduce un problema di sicurezza di un protocollo ad un problema di soddisfacibilità proposizionale (ovvero un'istanza di SAT). A partire dalla specifica IF del protocollo, infatti, il modulo genera automaticamente una formula proposizionale successivamente consegnata ad un risolutore SAT. In seguito, ogni soluzione restituita dal risolutore viene trasformata in un attacco al protocollo.

Oltre che per la rilevazione degli attacchi, infine, anche SATMC (come OFMC e CL-AtSe) può essere impiegato per la verifica della correttezza (per un numero limitato di sessioni) dei protocolli di rete.

1.2.4 TA4SP

Il modulo TA4SP (Tree Automata based on Automatic Approximations for the Analysis of Security Protocols)[12], invece, è in grado di verificare la correttezza dei protocolli di rete per un numero illimitato di sessioni. Questo avviene attraverso l'approssimazione delle conoscenze dell'attaccante mediante l'uso dei regular tree languages. Tale metodo permette di verificare se alcuni stati sono raggiungibili oppure no e, di conseguenza, se l'attaccante è in grado di venire a conoscenza di certi valori oppure no.

TA4SP può verificare se un protocollo ha una vulnerabilità (under approximation) oppure se è sicuro (over approximation). Nel caso in cui si stia procedendo alla stima delle conoscenze dell'attaccante mediante over-approximation e l'attaccante risulti in possesso di un valore che non dovrebbe essergli noto, invece, il modulo termina dichiarando che non può giungere a nessuna conclusione.

[11] A. Armando and L. Compagna. SAT-based Model-Checking for Security Protocols Analysis. Int. J. Inf. Sec., 7(1):3–32, 2008.
[12] Y. Boichut, P.-C. Héam, and O. Kouchnarenko. Tree Automata for Detecting Attacks on Protocols with Algebraic Cryptographic Primitives. Electr. Notes Theor. Comput. Sci., 239:57–72, 2009.

2 OpenID Connect

OpenID Connect è uno strato (layer) d'identità semplice basato sul protocollo OAuth 2.0. Esso consente ai Client di verificare l'identità di un Utente Finale sulla base della autenticazione effettuata da un Server di Autorizzazione, nonché di ottenere informazioni di base sul profilo dell'Utente Finale in maniera interoperabile e REST-like.

Le specifiche che lo descrivono si suddividono in più parti:

- **OpenID Connect Core** – Definisce il core delle funzionalità OpenID Connect: l'autenticazione basata sul protocollo OAuth 2.0 e l'uso dei claims per comunicare le informazioni relative all'End-User
- **OpenID Connect Discovery** – (Optional) Definisce come i Client possono trovare dinamicamente le informazioni relative agli OpenID Providers
- **OpenID Connect Dynamic Registration** – (Optional) Definisce come i Client si registrano dinamicamente con gli OpenID Providers

In questo lavoro è stata sviluppa solo la prima: **OpenID Connect Core 1.0**, ed in particolare il procedimento d'autenticazione denominato: **Authorization Code Flow**.

Di seguito sono definite le principali funzionalità dell'OpenID Connect: autenticazione basata su OAuth 2.0 e l'uso delle asserzioni per comunicare informazioni relative all'utente finale.

2.1 Introduzione

OpenID Connect è un strato d'identità semplice basato sul protocollo OAuth 2.0. Come background, le specifiche OAuth 2.0 Authorization Framework [13][RFC6749] e OAuth 2.0 Bearer Token Usage [14][RFC6750] forniscono un framework generale ad applicazioni di terze parti per ottenere ed usare accessi limitati a risorse HTTP. Essi definiscono i meccanismi per ottenere ed usare Access Token per accedere alle risorse, ma non definiscono metodi standard per fornire informazioni sull'identità. In particolare, senza il

[13] Hardt, D., "The OAuth 2.0 Authorization Framework," RFC 6749, October 2012
[14] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage," RFC 6750, October 2012

profiling di OAuth 2.0, è incapace di fornire informazioni relative all'autenticazione dell'utente finale.

OpenID Connect implementa l'autenticazione come un'estensione del processo di autorizzazione di OAuth 2.0. L'uso di questa estensione è richiesta dai Client per includere il valore della variabile **openid** nella richiesta di autenticazione. L'informazione relativa all'esecuzione dell'autenticazione è restituito come un JSON Web Token [15](JWT) chiamato ID Token. I Server di Autenticazione OAuth 2.0 che implementato OpenID Connect sono anche nominati OpenID Providers (OPs). I Client OAuth 2.0 che usano OpenID Connect sono anche denominati Relying Parties (RPs).

Questa specifica assume che il Relying Party abbia già ottenuto sufficienti credenziali ed abbia fornito informazioni necessarie per usare l'OpenID Provider, incluso il suo indirizzo dell'Authorization Endpoint e del Token Endpoint. Questa informazione è generalmente ottenuta attraverso la Discovery, come descritto nell'OpenID Connect Discovery 1.0[16], o può essere ottenuta attraverso altri meccanismi.

Allo stesso modo, questa specifica presuppone che il Relying Party abbia già ottenuto le sufficienti credenziali ed abbia fornito le informazioni necessarie per utilizzare l'OpendID provider. Questo avviene normalmente tramite la registrazione dinamica, come descritto nell'OpenID Connect Dynamic Client Registration 1.0[17], oppure possono essere ottenute attraverso altri meccanismi.

2.2 Panoramica

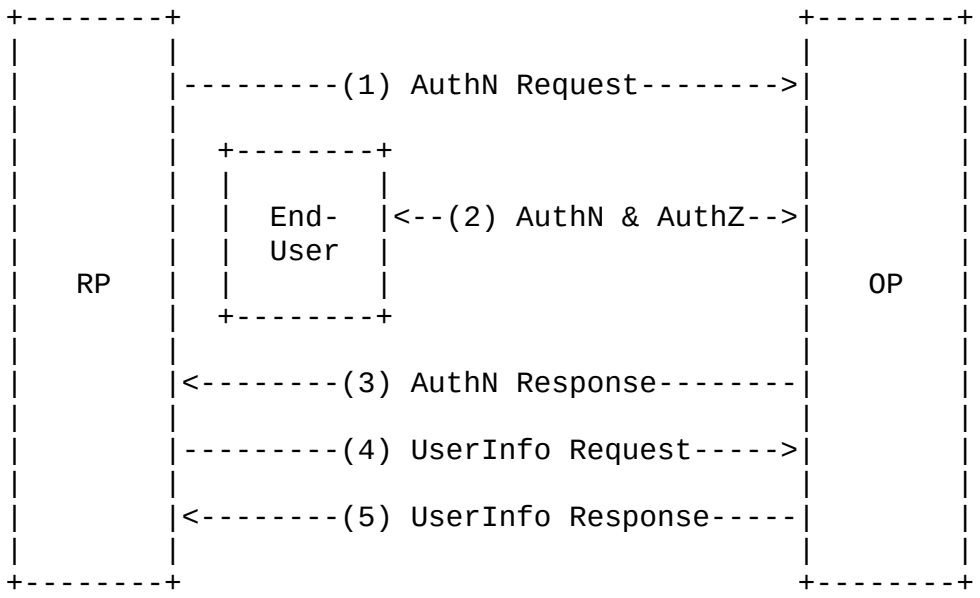
Il protocollo OpenID Connect, in astratto, segue i seguenti passaggi:

1. Il RP[18] (client) invia una richiesta al OpenID Provider[19] (OP).
2. L'OP autentica l'utente finale e ottiene l'autorizzazione.

[15] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)," draft-ietf-oauth-json-web-token (work in progress), November 2013
[16] Sakimura, N., Bradley, J., Jones, M., and E. Jay, "OpenID Connect Discovery 1.0," February 2014.
[17] Sakimura, N., Bradley, J., and M. Jones, "OpenID Connect Dynamic Client Registration 1.0," February 2014.
[18] Relying Party (RP): è un'applicazione client OAuth 2.0 che richiede l'autenticazione dell'utente finale e le asserzionio all'OpenId Provider.
[19] OpenID Provider (OP): è un server di autorizzazione OAuth 2.0 che è in grado di autenticare l'utente finale e fornire le asserzioni al Relyng Party in relazione all'evento di autenticazione e all'utente finale.

3. L'OP risponde con un ID Token[20] e di solito con un Access Token.
4. La RP può inviare una richiesta con l'Access Token all'UserInfo Endpoint.
5. Il UserInfo Endpoint[21] restituisce asserzioni relative all'utente finale.

Questi passaggi sono illustrati nel diagramma seguente:



[20] ID token: è un JSON Web Token (JWT) che contiene affermazioni relative all'evento di autenticazione. Esso può contenere altre entità.
[21] UserInfo Endpoint: e una risorsa protetta che, quando si presenta con un Access token da parte del Client, restituisce l'informazione autorizzata relativa all'utente finale rappresentato dal corrispondente Authorization Grant.

2.3 ID Token

La struttura dati dell'ID Token è la principale estensione che l'OpenID Connect fa all'OAuth 2.0 per abilitare l'Utente Finale ad essere autenticato. L'ID Token è un token di sicurezza che contiene asserzioni (claims) relative all'autenticazione di un Utente Finale, per un Authorization Server, quando si utilizza un Client. L'ID Token è rappresentato come un JSON Web Token (JWT).

Le seguenti asserzioni sono utilizzate dentro l'ID Token per tutti i flussi di OAuth 2.0 impiegati dall'OpenID Connect:

iss

OBBLIGATORIO. L'identificatore dell'emittente per l'Issuer della risposta. Il valore ISS è un URL case sensitive che utilizza lo schema https.

sub

OBBLIGATORIO. L'oggetto dell'identificatore. Un identificatore localmente unico e mai riassegnato all'interno dell'Emittente per l'Utente Finale, che è destinato ad essere consumato da parte del Client.

aud

OBBLIGATORIO. Destinatario (i) per il quale questo ID Token è indirizzato. Esso deve contenere l'OAuth 2.0 **client_id** del Relying Party come un valore audience. Esso può anche contenere identificatori di altri tipo audience. Nel caso generale, il valore *aud* è un array di stringhe case sensitive. Nel caso comune speciale comune quando c'è un solo pubblico, il valore di *aud* può essere una singola stringa case sensitive.

exp

OBBLIGATORIO. Il tempo di scadenza oltre il quale l'ID Token non deve essere accettato per l'elaborazione. L'elaborazione di questo parametro richiede che la data e l'ora corrente deve essere antecedente alla data e ora della scadenza indicata nel valore. Gli implementatori possono prevede un margine di tolleranza di pochi minuti che tenga conto dello scostamento degli orologi. Il valore è un JSON number che rappresenta il numero di secondi dal 1970-01-01T0:0:0Z calcolato in UTC fino alla data/ora.[22]

iat

[22] Per i dettagli riguardanti la data e l'ora vedere l'RFC 3339 ed in particolare UTC.

OBBLIGATORIO. L'ora in cui il JWT è stato rilasciato. Il suo valore è un JSON number.

auth_time

L'ora in cui è avvenuta l'autenticazione dell'Utente Finale. Il suo valore è un JSON number. Quando è fatta una richiesta *max_age* o quando *auth_time* è una richiesta come un asserzione essenziale, allora questo Claim è OBBLIGATORIO; in caso contrario, la sua inclusione è FACOLTATIVA.

nonce

Un valore stringa utilizzato per associare una sessione Client con un ID Token e per mitigare i replay attacks. Il valore è scambiato immodificato dall'Authentication Request all'ID Token. Se presente nell'ID Token, i Client devono verificare che il valore *nonce* è uguale al valore del parametro *nonce* inviato nell'Authentication Request. Se presente nell'Authentication Request, gli Authorization Server devono includere un claim con valore *nonce* nell'ID Token con il valore claim inviato nella richiesta di autenticazione. Gli Authorization Server non devono eseguire nessun altra elaborazione sui valori di *nonce* utilizzati. Il valore *nonce* è una stringa case sensitive.

acr

FACOLTATIVO. Riferimento all'Authentication Context Class. Una stringa che specifica un valore di riferimento alla classe di contesto di autenticazione che identifica l'esecuzione dell'autenticazione. Il valore *acr* è una stringa case sensitive.

amr

FACOLTATIVO. Riferimento ai metodi di autenticazione Un JSON array di stringhe che sono identificatori per i metodi di autenticazione utilizzati. Il valore *amr* è un'array di stringhe case sensitive.

azp

FACOLTATIVO. Soggetto autorizzato – l'entità al quale l'ID Token è stato rilasciato. Se presente deve contenere l'OAuth 2.0 Client ID dell'entità. Questa affermazione è necessaria solo quando l'ID Token ha un singolo valore di audience e quando l'audience è diverso dal soggetto autorizzato. Può essere incluso anche quando il soggetto autorizzato è lo stesso come unico audience. Il valore *azp* è una stringa case sensitive che contiene un valore StringOrURI.

Gli ID Token possono contenere altre entità. Eventuali Claims utilizzati che non sono compresi devono essere ignorati. Inoltre devono essere firmati utilizzando JWS[23] e facoltativamente possono essere sia firmati e

[23] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)," draft-ietf-jose-json-web-signature (work in progress), November 2013

successivamente criptati usando rispettivamente JWS e JWE[24], fornendo in tal modo l'autenticazione, l'integrità, il non ripudio e, facoltativamente, la riservatezza. Se l'ID Token è crittografato, deve essere firmato e poi cifrato con il risultato che deve essere in JWT nidificato, come definito in JWT. Gli ID Token non devono utilizzare *none* come il valore *alg* a meno che il tipo di risposta utilizzata ritorni senza ID Token dall'Authorization Endpoint (come ad esempio quando si utilizza il flusso Authorization Code) ed il Client esplicitamente ha richiesto l'uso di *none* al momento della registrazione.

Il seguente è un esempio del set di Claims (la JWT Claims Set) in un ID Token:

```
{
   "iss": "https://server.example.com",
   "sub": "24400320",
   "aud": "s6BhdRkqt3",
   "nonce": "n-0S6_WzA2Mj",
   "exp": 1311281970,
   "iat": 1311280970,
   "auth_time": 1311280969,
   "acr": "urn:mace:incommon:iap:silver"
  }
```

2.4 Autenticazione

OpenID Connect esegue l'autenticazione per loggare l'utente finale o per determinare se l'utente finale è già connesso. Egli restituisce il risultato dell'autenticazione effettuata dal Server al client in maniera sicura in modo tale che il client possa contare su di esso. Per questo motivo in questo caso il Client è chiamato Relying Party (RP).

Il risultato di autenticazione viene restituito sotto forma di un ID token. Esso contiene asserzioni che esprimono informazioni quali l'Emittente (Issuer) , il Subject Identifier, quando scade l'autenticazione, ecc.

L'autenticazione può seguire uno dei tre seguenti percorsi:

- Authorization Code Flow (*response_type=code*),
- Implicit Flow (*response_type=id_token token or response_type=id_token*),
- Hybrid Flow (usando altri valori tipo di risposta definiti nella OAuth 2.0 Multiple Response Type Encoding Practices).

[24] Jones, M., Rescorla, E., and J. Hildebrand, "JSON Web Encryption (JWE)," draft-ietf-jose-json-web-encryption (work in progress), November 2013

I flussi determinano come l'ID Token e l'Access Token vengono restituiti al Client.

Le caratteristiche dei tre flussi sono riassunti nella seguente tabella. La tabella è destinata a fornire alcune informazioni su quali flussi scegliere in particolari contesti.

Property	Authorization Code Flow	Implicit Flow	Hybrid Flow
All tokens returned from Authorization Endpoint	no	yes	no
All tokens returned from Token Endpoint	yes	no	no
Tokens not revealed to User Agent	yes	no	no
Client can be authenticated	yes	no	yes
Refresh Token possible	yes	no	yes
Communication in one round trip	no	yes	no
Most communication server-to-server	yes	no	varies

Il flusso utilizzato è determinato dal valore *response_type* contenuto nella Richiesta di Autorizzazione. Questi valori di *response_type* selezionano i seguenti flussi:

"response_type" value	**Flow**
code	Authorization Code Flow
id_token	Implicit Flow
id_token token	Implicit Flow
code id_token	Hybrid Flow
code token	Hybrid Flow
code id_token token	Hybrid Flow

Tutti i valori di Response Type tranne il tipo *code* che viene definito dall'OAuth 2.0, sono definiti nelle specifiche OAuth 2.0 Multiple Response Type Encoding Practices.

In questo studio è stato analizzato **l'Authorization Code Flow**.

2.4.1 Autenticazione tramite l'Authorization Code Flow

In questa sezione è descritto come eseguire l'autenticazione utilizzando l'Authorization Code Flow. Quando si utilizza questo flusso, tutti i token vengono restituiti dal Token Endpoint. L'Authorization Code Flow restituisce un codice di autorizzazione (*Authorization Code)* al Client, che può successivamente scambiarlo direttamente per un ID Token e un Access Token. Questo offre il vantaggio di non esporre alcun token all'User Agent. L'Authorization Server può anche autenticare il Client prima di scambiare l'Authorization Code per un Access Token. L'Authorization Code flow è adatto per i Client in grado di mantenere in modo sicuro un *Client Secret* tra loro e l'Authorization Server.

2.4.1.1 I passi dell'Authorization Code Flow

L'Authorization Code Flow passa attraverso le seguenti fasi:

1. Il Client prepara una Richiesta di Autenticazione (*Authentication Request)* contenete i parametri di richiesta desiderati.
2. Il Client invia la richiesta all'Authorization Server.
3. L'Authorization Server autentica l'End-User.
4. L'Authorization Server ottiene il consenso/autorizzazione dall'End-User.
5. L'Authorization Server invia la risposta dell'End-User al Client con un *Authorization Code*.
6. Il Client richiede una risposta usando *l'Authorization Code* al Token Endpoint.
7. Il Client riceve una risposta che contiene un *ID Token* e un *Access Token* nel corpo della risposta.
8. Il Client convalida *l'ID Token* e recupera il *Subject Identifier*[25] dell'End-User.

2.4.1.2 Authorization Endpoint

L'Authorization Endpoint esegue l'autenticazione dell'End-User. Ciò viene fatto con l'invio dell'User Agent all'Authorization Server's Authorization Endpoint per l'autenticazione e l'autorizzazione, utilizzando i parametri definiti dall'OAuth 2.0 ed i parametri aggiuntivi e i valori dei parametri definiti dall'OpenID Connect.
La comunicazione con Authorization Endpoint utilizza il protocollo TLS.

[25] E' un identificatore localmente unico e mai riassegnato all'interno dell'Emittente per l'End-User, il quale è destinao ad essere consumato da parte del Client.

2.4.1.2.1 Authentication Request

Una richiesta di autenticazione è un richiesta di autenticazione OAuth 2.0 che richiede che l'End-User sia autenticato dall'Authorization Server.

I Server di Autenticazione devono sostenere l'uso dei metodi HTTP GET e POST definiti nell'RFC 2616 presso l'Authorization Endpoint. I Client possono usare i metodi HTTP GET e POST per inviare la richiesta di autorizzazione al Server di Autenticazione. Se si usa il metodo http GET, i parametri della richiesta vengono serializzati mediante URI Query String Serialization. Se si utilizza il metodo http PUT, i parametri della richiesta vengono serializzati utilizzando il Form Serialization.

OpenID Connect utilizza i seguenti parametri per la richiesta OAuth 2.0 con l'Authorization Code Flow:

scope

OBBLIGATORIO. Le richieste di OpenID Connect devono contenere valorizzare il campo scope con *openid.* Se il valore di scope *openid* non è presente, il comportamento è completamente indeterminato. Altri valori di scope possono essere presenti. I valori di scope utilizzati che non sono riconosciuti da un'implementazione dovrebbero essere ignorati.

response_type

OBBLIGATORIO. Il valore dell'OAuth 2.0 Response Type che determina il flusso di elaborazione di autorizzazione da utilizzare, includendo quali parametri vengono restituiti dagli endpoint utilizzati. Quando si utilizza l'Authorization Code Flow questo valore è *code*.

client_id

OBBLIGATORIO. Un identificatore OAuth 2.0 Client valido per l'Authorization Server.

redirect_uri

OBBLIGATORIO. L'URI di Reindirizzamento a cui verrà inviata la risposta. Questo URI deve corrispondere esattamente a uno dei valori URI di reindirizzamento per il Client preregistrato presso il OpenID Provider, con l'abbinamento eseguita come descritto nella RFC3986 (Simple String Comparison). Quando si utilizza questo flusso, L'URI di Reindirizzamento deve utilizzare lo schema HTTPS; tuttavia, può utilizzare lo schema http, a condizione che il Client Type è confidenziale, e purchè l'OP consenta l'utilizzo di URI HTTP di .reindirizzamento. Il rendirizzamento può utilizzare un

sistema alternativo, come uno che ha lo scopo di identificare un callback in un'applicazione nativa.

state

CONSIGLIATO. Valore opaco utilizzato per mantenere lo stato tra la richiesta e il callback. Tipicamente, la mitigazione del Cross-Site Request Forgery (CSRF, XSRF) avviene vincolando crittograficamente il valore di questo parametro con un cookie del browser.

OpenID Connect utilizza anche il seguente parametro di richiesta OAuth 2.0, che è definito in OAuth 2.0 Multiple Response Type Encoding Practices.

response_mode

FACOLTATIVO. Informa Authorization Server del meccanismo da utilizzare per la restituzione dei parametri dall'Authorization Endpoint.. L'uso di questo parametro è sconsigliato quando la modalità di risposta che verrebbe richiesto è la modalità predefinita specificata per il tipo di risposta.

Questa specifica definisce anche i seguenti parametri di richiesta:

nonce

FACOLTATIVO. Valore Stringa utilizzato per associare una sessione Client con un ID Token, e per mitigare gli attacchi di replay. Il valore viene passato immodificato dalla richiesta di autorizzazione per l'ID Token. Nei valori *nonce* deve essere presente un'entropia sufficiente per impedire agli aggressori di indovinare i valori.

display

FACOLTATIVO. Valore stringa ASCII che specifica come l'Authorization Server visualizza le pagine dell'interfaccia utente di autenticazione e autorizzazione per l'utente finale. I valori definiti sono: page, popup, touch, wap.
L'Authorization Server può anche tentare di rilevare le capacità del User Agent e presentare un display appropriato.

prompt

FACOLTATIVO. Spazio delimitato, lista di stringhe ASCII case sensitive che specifica se l'Authorization Server richiede l'utente finale per la reautenticazione e autorizzazione. I valori definiti sono: none, login, consent, select_account,

max_age

FACOLTATIVO. La durata massima dell'autenticazione. Specifica il tempo consentito trascorso in secondi dall'ultima volta che

l'utente finale è stato attivamente autentico dall'OP. Se il tempo trascorso è maggiore di questo valore, l'OP deve cercare attivamente di ri-autenticare l'utente finale. (Il parametro di richiesta max_age corrisponde al PAPE parametro di richiesta OpenID 2.0 *max_auth_age*.) Quando si utilizza max_age, l'ID Token restituito deve comprendere un Valore Claim *auth_time*.

ui_locales

FACOLTATIVO. Il linguaggi preferito dall'End-User e gli script per l'interfaccia utente, rappresentate come una lista separata da spazi di BCP47 [RFC5646] valori delle variabili lingua, in ordine di preferenza.

id_token_hint

FACOLTATIVO. L'ID Token precedentemente rilasciato dall'Authorization Server viene passato come un suggerimento circa attuale o passata sessione autenticata dell'utente finale con il Cliente.

login_hint

FACOLTATIVO. Suggerisce all'Authorization Server sull'identificatore di login dell'utente finale (se necessario). Questo suggerimento può essere utilizzato da un RP se prima chiede l'utente finale per il suo indirizzo e-mail (o altro identificativo) e poi vuole passare tale valore come un suggerimento per il servizio di autorizzazione scoperto.

acr_values

FACOLTATIVO. Valore della Requested Authentication Context Class Reference. Stringa che specifica i valori ACR che il server di autorizzazione ha richiesta di utilizzare per l'elaborazione di questa richiesta di autenticazione, con i valori che appaiono in ordine di preferenza separato da spazi.

Il seguente è un esempio HTTP di risposta 302 reindirizzamento da parte del Cliente, che innesca la User Agent per fare una richiesta di autenticazione per l'Authorization Endpoint:

```
HTTP/1.1 302 Found
  Location: https://server.example.com/authorize?
    response_type=code
    &scope=openid%20profile%20email
    &client_id=s6BhdRkqt3
    &state=af0ifjsldkj

&redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
```

2.4.1.2.2 Authentication Request Validation

L'Authorization Server deve convalidare la richiesta ricevuta come segue:

1. L'Authorization Server deve convalidare tutti i parametri OAuth 2.0 secondo la specifica OAuth 2.0.
2. Verificare che un parametro di scope è presente e contiene il valore *OpenID*. (Se nessun valore di scope OpenID è presente, la richiesta può essere ancora una richiesta OAuth 2.0 valido, ma non è una richiesta di OpenID Connect.)
3. L'Authorization Server deve verificare che tutti i parametri richiesti sono presenti e il loro uso è conforme a questa specifica.
4. Se è richiesto il sub (soggetto) Claim con un valore specifico per l'ID Token, il server di autorizzazione deve inviare solo una risposta positiva se l'utente finale identificato da quel valore sub ha una sessione attiva con il server di autorizzazione o è stata autenticata come a seguito della richiesta. Il server di autorizzazione non deve rispondere con un ID Token o Access Token per un utente diverso, anche se hanno una sessione attiva con l'Authorization Server.

Come specificato nell'OAuth 2.0 [RFC6749], gli Authorization Server dovrebbero ignorare i parametri di richiesta non riconosciuti.

Se l'Authorization Server rileva qualche errore, deve restituire una risposta di errore.

2.4.1.2.3 Authorization Server Authenticates End-User

Se la richiesta è valida, il server di autorizzazione tenta di autenticare l'utente finale o determina se l'utente finale è autenticato, a seconda dei valori dei parametri di richiesta utilizzati. I metodi utilizzati dal server delle autorizzazioni per autenticare l'utente finale (ad esempio nome utente e password, cookie di sessione, ecc) sono al di là del campo di applicazione del presente disciplinare. Un'interfaccia utente di autenticazione può essere visualizzato dal server di autorizzazione, a seconda dei valori dei parametri di richiesta utilizzati ed i metodi di autenticazione utilizzati.

Il server di autorizzazione deve tentare di autenticare l'utente finale nei seguenti casi:

- L'utente finale non è già autenticato.
- La richiesta di autenticazione contiene il parametro di richiesta con il valore login. In questo caso, il server di autorizzazione deve

autenticare nuovamente l'utente finale, anche se l'utente finale è già autenticato.

Il server di autorizzazione non deve interagire con l'utente finale nel seguente caso:

- La richiesta di autenticazione contiene il parametro di richiesta con il valore none. In questo caso, il server di autorizzazione deve restituire un errore se l'utente finale non è già autenticato o non può essere in silenzio autenticati.

Quando si interagisce con l'utente finale, il server di autorizzazione deve assumere misure adeguate nei confronti di Cross-Site Request Forgery e Clickjacking come, descritto nei paragrafi 10.12 e 10.13 di OAuth 2.0 [RFC6749].

2.4.1.2.4 Authorization Server Obtains End-User Consent/Authorization

Una volta che l'utente finale è autenticato, il server di autorizzazione deve ottenere una decisione di autorizzazione prima di rilasciare le informazioni al Relying Party. Quando consentito dai parametri di richiesta utilizzati, questo può essere fatto attraverso un dialogo interattivo con l'utente finale che rende chiaro ciò che viene consentito o stabilire il consenso attraverso le condizioni per l'elaborazione della richiesta o di altri mezzi.

2.4.1.2.5 Successful Authentication Response

Una risposta di autenticazione è un messaggio di risposta d'autorizzazione OAuth 2.0 restituito dall'OP's Authorization Endpoint in risposta al messaggio di richiesta di autorizzazione inviato dal Relying Party (RP).

Quando si utilizza l'Authorization Code Flow, la risposta di autorizzazione deve riportare i parametri definiti nella sezione 4.1.2 di OAuth 2.0 [RFC6749].

Il seguente è un esempio di risposta positiva con questo flusso:

```
HTTP/1.1 302 Found
  Location: https://client.example.org/cb?
    code=SplxlOBeZQQYbYS6WxSbIA
    &state=af0ifjsldkj
```

2.4.1.2.6 Authentication Error Response

Una risposta di errore di autenticazione è un messaggio OAuth 2.0 Authorization Error Response restituito dall'OP's Authorization Endpoint in risposta al messaggio di richiesta di autorizzazione inviato dal Relying Party (RP).

Se l'utente finale nega la richiesta oppure l'autenticazione dell'utente finale non riesce, l'OP (Authorization Server) informa il RP (Client) utilizzando i parametri di risposta di errore di cui alla sezione 4.1.2.1 di OAuth 2.0 [RFC6749]. (Errori HTTP estranei a RFC 6749 vengono restituiti alla User Agent utilizzando il codice di stato HTTP appropriato.)

A meno che il reindirizzamento URI non sia valido, l'Authorization Server restituisce una risposta di errore al Cliente, al URI di reindirizzamento specificato nella richiesta di autorizzazione, con i parametri di stato error. Altri parametri non devono essere restituiti.

Oltre ai codici di errore definiti nella Sezione 4.1.2.1 di OAuth 2.0, questa specifica definisce anche i seguenti codici di errore:

- interaction_required
- login_required
- account_selection_required
- consent_required
- invalid_request_uri
- invalid_request_object
- request_not_supported
- request_uri_not_supported
- registration_not_supported

I parametri di risposta d'errore sono i seguenti:

errore
: RICHIESTO. Codice di errore.

error_description
: FACOLTATIVO. Testo ASCII codificato leggibile dell'errore.

error_uri
: FACOLTATIVO. URI di una pagina web che contiene ulteriori informazioni sull'errore.

state
: OAuth valore di stato 2.0. Obbligatorio se la richiesta di autorizzazione comprendeva il parametro di stato. Impostare il valore ricevuto dal client.

Quando si utilizza l'Authorization Code Flow, i parametri di risposta di errore vengono aggiunti alla componente di query del reindirizzamento URI, a meno che non è stata specificata una modalità di risposta diversa.

La seguente è un esempio di risposta di errore utilizzando questo flusso:

```
HTTP/1.1 302 Found
Location: https://client.example.org/cb?
  error=invalid_request
  &error_description=
    Unsupported%20response_type%20value
  &state=af0ifjsldkj
```

2.4.1.2.7 Authentication Response Validation

Quando si utilizza l'Authorization Code Flow, il Client deve convalidare la risposta secondo la RFC 6749.

2.4.1.3 Token Endpoint

Per ottenere un Access Token, un ID Token, e facoltativamente un Refresh Token, l'RP (Client) invia un richiesta di token al Token Endpoint per ottenere un token di risposta, come descritto nella Sezione 3.2 di OAuth 2.0 [RFC6749], quando si utilizza l'Authorization Code Flow.

La comunicazione con il Token Endpoint deve utilizzare il protocollo TLS.

2.4.1.3.1 Token Request

Un client effettua una richiesta di Token presentando la sua Authorization Gran (sotto forma di un *Authorization Code*) al Token Endpoint utilizzando il valore *grant_type=authorization_code*, come descritto nella Sezione 4.1.3 di OAuth 2.0 [RFC6749]. Se il client è un Confidential Client, allora deve autenticarsi al Token Endpoint utilizzando il metodo di autenticazione registrato per la sua *client_id*.

Il Client invia i parametri al Token Endpoint utilizzando il metodo HTTP POST e il modulo di serializzazione come descritto nella Sezione 4.1.3 di OAuth 2.0 [RFC6749].

Il seguente è un esempio di una richiesta di Token:

```
POST /token HTTP/1.1
  Host: server.example.com
  Content-Type: application/x-www-form-urlencoded
  Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW


grant_type=authorization_code&code=SplxlOBeZQQYbYS6WxS
bIA&redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
```

2.4.1.3.2 Token Request Validation

L'Authorization Server deve convalidare il Token di richiesta come segue:

- Autenticare il Client se è stato emesso Client Credentials o se si utilizza un altro metodo di autenticazione del client.
- Assicurarsi che l'Authorization Code è stato rilasciato al Client autenticato.
- Verificare che l'Authorization Code sia valido.
- Se possibile, verificare che l'Authorization Code non sia stato utilizzato in precedenza.
- Assicurarsi che il valore del parametro *redirect_uri* è identico al valore del parametro *redirect_uri* che è stato incluso nella richiesta di autorizzazione iniziale. Se il valore del parametro *redirect_uri* non è presente, quando c'è un solo valore *redirect_uri* registrato, l'Authorization Server può restituire un errore (dato che il Client avrebbe dovuto includere il parametro) o può procedere senza un errore (dato che OAuth 2.0 permette l'omissione del parametro).
- Verificare che l'Authorization Code utilizzato sia stato rilasciato in risposta a una richiesta di autenticazione OpenID Connect (in modo che l'ID Token venga restituito dal Token Endpoint).

2.4.1.3.3 Successful Token Response

Dopo aver ricevuto e validato una Richiesta di Token dal Client, valida ed autorizzata, l'Authorization Server restituisce una risposta di successo che include un ID Token e un Access Token. I parametri nella risposta di successo sono definite nella sezione 4.1.4 di OAuth 2.0 [RFC6749]. La risposta utilizza il tipo di supporto application / json.

L'OAuth 2.0 token_type response parameter deve essere *Bearer,* come specificato nella OAuth 2.0 Bearer Token Usage [RFC6750], a meno che non sia stato negoziato un altro tipo di token con il Client. Il Server dovrebbe

sostenere il Bearer Token Type; l'uso di altri tipi di token è al di fuori del campo di applicazione della presente specifica.

Oltre ai parametri di risposta specificati da OAuth 2.0 devono essere inclusi i seguenti parametri nella risposta:

id_token
Valore d'ID token associato alla sessione autenticata.

Tutte le Token Response che contengono Token, Segreti, o altre informazioni sensibili devono includere i seguenti campi e valori di intestazione di risposta HTTP:

Header Name	Header Value
Cache-Control	no-store
Pragma	no-cache

Il seguente è un esempio di una Successful Token Response.

```
  HTTP/1.1 200 OK
  Content-Type: application/json
  Cache-Control: no-store
  Pragma: no-cache

  {
   "access_token": "SlAV32hkKG",
   "token_type": "Bearer",
   "refresh_token": "8xLOxBtZp8",
   "expires_in": 3600,
   "id_token":
"eyJhbGciOiJSUzI1NiIsImtpZCI6IjFlOWdkazcifQ.ewogImlzc
yI6ICJodHRwOi8vc2VydmVyLmV4YW1wbGUuY29tIiwKICJzdWIiOiAiMjQ4M
jg5NzYxMDAxIiwKICJhdWQiOiAiczZCaGRSa3F0MyIsCiAibm9uY2UiOiAib
i0wUzZfV3pBMk1qIiwKICJleHAiOiAxMzExMjgxOTcwLAogImlhdCI6IDEzM
TEyODA5NzAKfQ.ggW8hZ1EuVLuxNuuIJKX_V8a_OMXzR0EHR9R6jgdqrOOF4
daGU96Sr_P6qJp6IcmD3HP99Obi1PRs-cwh3LO-
p146waJ8IhehcwL7F09JdijmBqkvPeB2T9CJNqeGpe-
gccMg4vfKjkM8FcGvnzZUN4_KSP0aAp1tOJ1zZwgjxqGByKHiOtX7Tpd
QyHE5lcMiKPXfEIQILVq0pc_E2DzL7emopWoaoZTF_m0_N0YzFC6g6EJbOEo
RoSK5hoDalrcvRYLSrQAZZKflyuVCyixEoV9GfNQC3_osjzw2PAithfubEEB
LuVVk4XUVrWOLrLl0nx7RkKU8NXNHq-rvKMzqg"
  }
```

Come specificato nell'OAuth 2.0 [RFC6749], i Client dovrebbero ignorare i parametri di risposta non riconosciuti.

2.4.1.3.4 Token Error Response

Se il Token Request non è valido o non autorizzato, l'Authorization Server restituisce una risposta di errore. I parametri della risposta di errore sono definiti come nella sezione 5.2 di OAuth 2.0 [RFC6749]. Il corpo della risposta HTTP utilizza il tipo di supporto application / json con codice di risposta HTTP 400.

Il seguente è un esempio di errore di risposta da Token:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Cache-Control: no-store
Pragma: no-cache

{
 "error": "invalid_request"
}
```

2.4.1.3.5 Token Response Validation

Il Client deve convalidare la risposta Token come segue:

1. Seguire le regole di convalida in RFC 6749, in particolare quelli nelle sezioni 5.1 e 10.12.
2. Seguire le regole di convalida ID Token.
3. Seguire le regole di convalida di Access Token.

2.4.1.3.6 ID Token

Il contenuto dell'ID Token sono descritto nella sezione 2.3. Quando si utilizza il Authorization Code Flow, si applicano i seguenti requisiti aggiuntivi all'ID Token Claims:

at_hash

FACOLTATIVO. Il valore hash dell'Access Token. Il suo valore è la codifica base64url della metà a sinistra degli ottetti della rappresentazione ASCII del valore ac*cess_tok*en, dove l'algoritmo di hash utilizzato è l'algoritmo hash utilizzato nel parametro alg del'ID Token JWS [JWS].

2.4.1.3.7 ID Token Validation

I client devono convalidare l'ID Token nella Token Response nel seguente modo:

1. Se l'ID Token è crittografato, decifra usando le keys e gli algoritmi che il Client ha specificato durante la registrazione, lo stesso che l'OP ha utilizzato per crittografare l'ID Token. Se la crittografia è stata negoziata con l'OP in fase di registrazione e l'ID Token non è crittografato, la RP dovrebbe respingerla.
2. L'Identifier Issuer per l'OpenID Provider (che in genere è ottenuto durante la Discovery) deve corrispondere esattamente al valore della *ISS* (emittente) Claim.
3. Il Client deve convalidare che il Claim *aud* (audience) contienga il valore *client_id* registrato presso l'Emittente individuato dal Claim *ISS* (emittente) come audience. L'aud (audience) Claim può contenere un array con più di un elemento. L'ID Token deve essere respinto se l'ID Token non elenca il Client come un audience valido, o se contiene audince supplementari non attendibile dal Client.
4. Se l'ID Token contiene più audience, il Client deve verificare che un Claim *azp* sia presente.
5. Se un *AZP* (soggetto autorizzato) è presente, il Client deve verificare che la sua *client_id* è il valore del Claim.
6. Se l'ID Token viene ricevuto tramite la comunicazione diretta tra il Client e il Token Endpoint (come in questo flusso), la convalida del server TLS può essere utilizzato per convalidare l'emittente al posto di controllare la firma del token. Il Client deve convalidare la firma di tutti gli altri ID Tokens secondo JWS [JWS] utilizzando l'algoritmo specificato nel parametro nell'intestazione JWT *alg*. Il Client deve utilizzare le chiavi fornite dall'Emittente.
7. Il valore *alg* dovrebbe essere di default RS256 o l'algoritmo inviato dal Client nel parametro *id_token_signed_response_alg* durante la registrazione.
8. Se il parametro di intestazione *alg* JWT utilizza un algoritmo basato su MAC come HS256, HS384, HS512 o, gli ottetti della rappresentazione UTF-8 della *client_secret* corrispondente al *client_id* contenute nelle *AUD* (audience) i Claim sono utilizzati come chiave per convalidare la firma. Per gli algoritmi basati su MAC, il comportamento non è specificato se l'aud è multivalore o se un valore azp è presente diverso rispetto al valore *aud*.
9. L'ora corrente deve essere prima dell'ora rappresentata dal Claim *exp*.
10. Il Claim *iat* può essere utilizzato per respingere i token emessi troppo distanti dal tempo corrente, limitando la quantità di tempo che i

2. Il Client invia la richiesta all'Authorization Server.
3. L'Authorization Server autentica l'utente finale.
4. L'Authorization Server ottiene il consenso e l'autorizzazione dell'utente finale.
5. L'Authorization Server invia la risposta dell'utente finale al Cliente con un ID Token e, se richiesto, un Acces Token.
6. Il Client convalida l'ID Token e recupera Oggetto Identificatore dell'utente finale.

2.4.3 Autenticazione usando l'Hybrid Flow

Quando si utilizza l'Hybrid Flow, alcuni token vengono restituiti dall'Authorization Endpoint e gli altri vengono restituiti dal Token Endpoint. I meccanismi per la restituzione del token nel Hybrid Flow sono specificati in OAuth 2.0 Multiple Response Type Encoding Practices.

2.4.3.1 Hybrid Flow Steps

In questo studio l'implementazione non verrà esaminata.

L'Hybrid Flow segue le seguenti fasi:

1. Il Client prepara una richiesta di autenticazione contenente i parametri di richiesta desiderati.
2. Il Client invia la richiesta all'Authorization Server.
3. L'Authorization Server autentica l'utente finale.
4. L'Authorization Server ottiene il consenso e l'autorizzazione dall'utente finale.
5. L''Authorization Server invia la risposta dell'utente finale al Cliente con un Authorization Code e, a seconda del tipo di risposta, uno o più parametri aggiuntivi.
6. Il Client richiede una risposta utilizzando l'Authorization Code al Token Endpoint.
7. Il Client riceve una risposta che contiene un ID Token ed un Access Token nel corpo della risposta.
8. Il Client convalida l'ID Token e recupera l'Oggetto Identificatore dell'utente finale.

2.5 Claims

In questa sezione viene descritto il modo un cui il Client può ottenere le asserzioni (Claims) relative all'utente finale ed all'evento di autenticazione. Si definisce anche un insieme standard di asserzioni. Un set predefinito di Claim può essere richiesto utilizzando valori specifici di scope, ovvero possono essere richiesti singoli Claim utilizzando il claims request parameter. Il Claim può provenire direttamente dal OpenID provider o da fonti distribuite.

2.5.1 Standard Claims

Di seguito si definiscono un insieme standard di Claim. Possono essere richiesti per essere restituite sia nella risposta UserInfo o nell'ID Token.

Member	Type	Description
sub	string	Subject - Identifier for the End-User at the Issuer.
name	string	End-User's full name in displayable form including all name parts, possibly including titles and suffixes, ordered according to the End-User's locale and preferences.
given_name	string	Given name(s) or first name(s) of the End-User. Note that in some cultures, people can have multiple given names; all can be present, with the names being separated by space characters.
family_name	string	Surname(s) or last name(s) of the End-User. Note that in some cultures, people can have multiple family names or no family name; all can be present, with the names being separated by space characters.
middle_name	string	Middle name(s) of the End-User. Note that in some cultures, people can have multiple middle names; all can be present, with the names being separated by space characters. Also note that in some cultures, middle names are not used.
nickname	string	Casual name of the End-User that

		may or may not be the same as the given_name. For instance, a nicknamevalue of Mike might be returned alongside agiven_name value of Michael.
preferred_username	string	Shorthand name by which the End-User wishes to be referred to at the RP, such as janedoe or j.doe. This value MAY be any valid JSON string including special characters such as @, /, or whitespace. The RP MUST NOT rely upon this value being unique, as discussed in**Section 5.7**.
profile	string	URL of the End-User's profile page. The contents of this Web page SHOULD be about the End-User.
picture	string	URL of the End-User's profile picture. This URL MUST refer to an image file (for example, a PNG, JPEG, or GIF image file), rather than to a Web page containing an image. Note that this URL SHOULD specifically reference a profile photo of the End-User suitable for displaying when describing the End-User, rather than an arbitrary photo taken by the End-User.
website	string	URL of the End-User's Web page or blog. This Web page SHOULD contain information published by the End-User or an organization that the End-User is affiliated with.
email	string	End-User's preferred e-mail address. Its value MUST conform to the **RFC 5322** [RFC5322] addr-spec syntax. The RP MUST NOT rely upon this value being unique, as discussed in **Section 5.7**.
email_verified	boolean	True if the End-User's e-mail address has been verified; otherwise false. When this Claim Value istrue, this means that

		the OP took affirmative steps to ensure that this e-mail address was controlled by the End-User at the time the verification was performed. The means by which an e-mail address is verified is context-specific, and dependent upon the trust framework or contractual agreements within which the parties are operating.
gender	string	End-User's gender. Values defined by this specification are female and male. Other values MAY be used when neither of the defined values are applicable.
birthdate	string	End-User's birthday, represented as an **ISO 8601:2004** [ISO8601-2004] YYYY-MM-DD format. The year MAY be 0000, indicating that it is omitted. To represent only the year, YYYY format is allowed. Note that depending on the underlying platform's date related function, providing just year can result in varying month and day, so the implementers need to take this factor into account to correctly process the dates.
zoneinfo	string	String from zoneinfo **[zoneinfo]** time zone database representing the End-User's time zone. For example,Europe/Paris or America/Los_Angeles.
locale	string	End-User's locale, represented as a **BCP47** [RFC5646] language tag. This is typically an **ISO 639-1 Alpha-2**[ISO639-1] language code in lowercase and an **ISO 3166-1 Alpha-2** [ISO3166-1] country code in uppercase, separated by a dash. For example, en-US orfr-CA. As a compatibility note, some implementations have used an

		underscore as the separator rather than a dash, for example, en_US; Relying Parties MAY choose to accept this locale syntax as well.
phone_number	string	End-User's preferred telephone number. **E.164** [E.164] is RECOMMENDED as the format of this Claim, for example, +1 (425) 555-1212 or +56 (2) 687 2400. If the phone number contains an extension, it is RECOMMENDED that the extension be represented using the **RFC 3966** [RFC3966] extension syntax, for example, +1 (604) 555-1234;ext=5678.
phone_number_verified	boolean	True if the End-User's phone number has been verified; otherwise false. When this Claim Value is true, this means that the OP took affirmative steps to ensure that this phone number was controlled by the End-User at the time the verification was performed. The means by which a phone number is verified is context-specific, and dependent upon the trust framework or contractual agreements within which the parties are operating. When true, the phone_number Claim MUST be in E.164 format and any extensions MUST be represented in RFC 3966 format.
address	JSON object	End-User's preferred postal address. The value of theaddress member is a JSON **[RFC4627]** structure containing some or all of the members defined in**Section 5.1.1**.
updated_at	number	Time the End-User's information was last updated. Its value is a JSON number representing the number of seconds from 1970-01-01T0:0:0Z as measured in UTC until the date/time.

2.5.2 UserInfo Endpoint

L'UserInfo Endpoint è un OAuth 2.0 Protected Resource che restituisce Claims relativi all'Utente Finale autenticato. Per ottenere le asserzioni richieste relative all'End-User, il Client fa una richiesta all'UserInfo Endpoint usando un Access Token ottenuto attraverso l'autenticazione OpenID Connect. Queste asserzioni sono normalmente rappresentate da un oggetto JSON che contiene una raccolta di coppie nome e valore di Claims.

La comunicazione con il UserInfo endpoint deve utilizzare il protocollo TLS.

L'UserInfo Endpoint deve sostenere l'uso dei metodi HTTP GET e POST HTTP definiti nella RFC 2616.

L'UserInfo Endpoint deve accettare Access Tokens come definiti in OAuth 2.0 Bearer Token Usage [RFC6750].

2.5.2.1 UserInfo Request

Il Client invia la richiesta UserInfo utilizzando HTTP GET o HTTP POST. L'Access Token, ottenuto da una richiesta di autenticazione OpenID Connect, deve essere inviato come Bearer Token, come descritto alla Sezione 2 di OAuth 2.0 Bearer Token Usage [RFC6750].

Si raccomanda che la richiesta utilizzi il metodo GET HTTP e che l'Access Token sia inviato utilizzando il campo di intestazione dell'autorizzazione.

Il seguente è un esempio di una richiesta UserInfo:

```
GET /userinfo HTTP/1.1
  Host: server.example.com
  Authorization: Bearer SlAV32hkKG
```

2.5.2.2 Successful UserInfo Response

L'UserInfo Claims devono essere restituiti come membri di un oggetto JSON a meno che sia stata richiesta una risposta firmata o crittografata durante la registrazione del Client. I claims di cui al punto 2.5.1 possono essere restituiti, così come ulteriori claim non qui specificati.

Per motivi di privacy, gli OpenID provider possono scegliere di non restituire i valori per alcuni claim richiesti.

Se un claim non viene restituito, quel nome di Claim dovrebbe essere omesso dall'oggetto JSON che rappresenta l'entità; lo stesso non dovrebbe essere presente con un valore di stringa nulla o vuota.

Il Claim sub (soggetto) deve sempre essere restituito nella risposta UserInfo.

A causa della possibilità di attacchi di sostituzione del token, la risposta UserInfo non è garantita per essere relativa all'End-User identificato dall'elemento sub (soggetto) della ID Token. Il sub Claim nella risposta UserInfo deve essere verificato per corrispondere esattamente al sub Claim nell'ID Token; Se non corrispondono, non devono essere utilizzati i valori della risposta UserInfo.

Al ricevimento della richiesta UserInfo, l'UserInfo Endpoint deve restituire la serializzazione JSON della risposta UserInfo nel corpo della risposta HTTP a meno che non è stato specificato un formato diverso durante la registrazione [OpenID.Registration]. L'UserInfo Endpoint deve restituire un content-type header per indicare quale formato viene restituito. Il tipo di contenuto della risposta HTTP deve essere application / json se il corpo della risposta è un oggetto JSON; il corpo della risposta dovrebbe essere codificato usando UTF-8.

Se la risposta UserInfo è firmata e / o crittografata, ogni asserzione viene restituita in un JWT ed il content-type deve essere un application / JWT. La risposta può essere codificata anche senza essere firmata. Se vengono richiesti sia la firma che la crittografia, la risposta deve essere prima firmata e poi cifrata, con il risultato di essere un nidificato JWT.

Se firmata, la risposta UserInfo deve contenere il Claims *iss* (issuer) e *aud* (audience) in qualità di membri. Il valore *iss* dovrebbe essere del OP Emittente Identifier URL. Il valore *aud* dovrebbe essere o includere il valore RP's Client ID.

Il seguente è un esempio di una risposta UserInfo:

```
HTTP/1.1 200 OK
  Content-Type: application/json

  {
   "sub": "248289761001",
   "name": "Jane Doe",
   "given_name": "Jane",
   "family_name": "Doe",
   "preferred_username": "j.doe",
```

```
 "email": "janedoe@example.com",
 "picture": "http://example.com/janedoe/me.jpg"
}
```

2.5.2.3 UserInfo Error Response

Quando si verifica una condizione di errore, l'UserInfo Endpoint restituisce una risposta di errore come definito nella Sezione 3 di OAuth 2.0 Bearer Token Usage [RFC6750].

Il seguente è un esempio di una risposta di errore UserInfo:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: error="invalid_token",
  error_description="The Access Token expired"
```

2.5.2.4 UserInfo Response Validation

Il Client deve convalidare la risposta UserInfo come segue:

1. Verificare che l'OP ha risposto coincida con l'OP previsto attraverso un controllo del certificato del server TLS, per RFC 6125.
2. Se il Client ha fornito un parametro *userinfo_encrypted_response_alg* durante la registrazione, decifrare la risposta UserInfo utilizzando le chiavi specificate durante la registrazione.
3. Se la risposta è stata firmata, il Client deve convalidare la firma secondo JWS.

2.6 Considerazioni sulla sicurezza

Le specifiche di sicurezza dell'OpenID Connect fanno riferimento alle considerazioni definite nella sezione 10 dell'OAuth 2.0 [RFC6749] e nella sezione 5 dell'OAuth 2.0 Bearer Token Usage [RFC6750] a cui si rimanda per approfondimento. Inoltre, la specifica OAuth 2.0 Threat Model and Security Considerations [RFC6819] fornisce un ampio elenco di minacce e controlli che si applicano anche a questa specifica, dato che si basa sull'OAuth 2.0. Anche la ISO/IEC 29115 [ISO29115] fornisce rischi e contromisure che gli implementatori devono prendere in considerazione. Questi ultimi sono fortemente inviati a leggere attentamente i riferimenti in dettaglio e ad applicare le contromisure descritte.

Di seguito è riportato un elenco di attachi e contromisure previste nell'OpenID Connect.

Request Disclosure

Se non si prendono contromisure adeguate, una richiesta di informazioni può essere comunicata ad un attaccante, rappresentando una minaccia alla sicurezza ed alla privacy.

In aggiunta a quanto indicato nella sezione 5.1.1 della RFC6819, questo standard fornisce una via per garantire la confidenzialità della richiesta attraverso l'uso dei parametri request o request_uri, il cui contenuto della richiesta è criptato con un opportuno uso di chiavi e algoritmi di cifratura JWT. Ciò protegge anche nel caso di richiesta indiretta attraverso User Agent compromessi.

Server Masquerading

Un server malevolo potrebbe mascherarsi come server leggittimo utilizzando vari strategie. Per evitare questo tipo di attacco, il Client deve autenticare il Server.

In aggiunta a quanto indicato nella sezione 5.1.2 della RFC6819, questo standard fornisce un modo per autenticare il Sever sia attraverso l'uso di JWT firmati e crittografati con un opportuno uso di chiavi e algoritmi di cifratura.

Token Manufacture/Modification

Un utente malintenzionato potrebbe generare un token falso i modificare il contenuto del token (come i valori dei Claim o la firma) di un token esistente, procando l'errata concessione di accesso al Client. Per esempio, un Attacante potrebbe modificare il token da interpretare per estendere il periodo di validità; un Client potrebbe modificare il token per avere accesso alle informazioni che non dovrebbe essere in grado di visualizzare.

Ci sono due modi per mitigare questo attacco:

- Il token può essere firmato digitalmente dall'OP. Il Relying Party dovrebbe validare la firma digitale per verificare che è stato rilasciato dell'OP leggittima.
- Il token può essere mandato attraverso un canale protetto come TLS. In questa specifica il token è sempre inviato su un canale protetto del tipo TLS. Si noti, tuttavia, che questa contromisura è solo una difesa

contro attacchi di terze parti e non è applicabile nel caso in cui l'attaccante è il Client.

Access Token Disclosure

Gli Access Token sono credenziali utilizzate per accedere alle risorse protette, come definito nella sezione 1.4 of OAuth 2.0 [RFC6749]. Gli Access Token rappresentano un'autorizzazione di un End-User e non devono essere esposte a persone non autorizzate.

Server Response Disclosure

Le risposte del Server potrebbero contenere dati di autenticazione e i claim che includono informazioni sensibili del Client. La divulgazione dei contenuti di risposta può rendere vulnerabile ad altri tipo di attacchi.

La Server Response Disclosure può essere mitigata nei seguenti due modi:

- Se si usa il Response Type *code*. La risposta è inviata su un canale protetto TLS, dove il Client è autenticato dal *client_id* e *client_secret*.

Per gli altri Response Type, la risposta firmata può essere criptata con la chiave pubblica del Client o un segreto condiviso come JWT criptata con un opportuno uso di chiavi e algoritmi di cifratura.

Server Response Repudiation

Una risposta potrebbe essere ripudiata dal Server se i meccanismi non sono adeguati. Per esempio, se un Server non firma digitalmente una risposta, il Server può asserire che non è stata generata attraverso i servizi del Server.

Per mitigare questa minaccia, la risposa può essere firmata digitalmente dal Server utilizzando una chiave che supporti il non ripudio. Il Client deve convalidare la firma digitale per verificare che è stato rilasciato da un Server leggittimo e la sua integrità sia intatta.

Request Repudiation

Dal momento che è possibile per un Client compromesso o malevolo mandare una richiesta alla parte sbagliata, un Client che è stato autenticato utilizzando solo un bearer token può ripudiare qualsiasi transazione.

Per mitigare questa minaccia, il Server può richiede che la richiesta sia firmata digitalmente dal Client utilizzando una chiave che supporta il non ripudio. Il Server deve convalidare la firma digitale per verificare che è stato rilasciato da un Client e l'integrità è intatta.

Access Token Redirect

Un Attaccante utilizza un Access Token generato per una risorse per ottenere l'accesso a una seconda risorsa.

Per mitigare questo tipo di minaccia, l'Access Token dovrebbe avere audience e scope limitati. Un modo per implementarlo è di includere l'identifcatore della risorsa per cui è stato generato come audience. La risorsa verifica che i token in arrivo includano il suo identificatore come audience del token.

Token Reuse

Un attaccante tenta di utilizzare un one-time token, come un Authorization Code che è già stato utilizzato una volta sola con la risorsa prevista. Per mitigare questa minaccia, il token dovrebbe includere un timestamp e un valori di vita per assicurare che il token sia attualmente valido.

Alternativamente, il server può registrare lo stato di utilizzo del token e controllare lo stato di ogni richiesta.

Eavesdropping or Leaking Authorization Codes (Secondary Authenticator Capture)

Oltre ai modelli di attacco di cui alla Sezione 4.4.1.1 del RFC6819, un Authorization Code può essere catturato nell'User Agent in cui la sessione TLS è terminata se l'User Agent è infettatto da un malware. Tuttavia, la cattura non è utile fintanto che viene utilizzato sia il Client Authentication o una risposta criptata.

Token Substitution

La sostituzione del token è una classe di attacchi in cui un utente malintenzionato scambia viari token, compreso lo scambio di un Authorization Code per un utente legittimo con un altro token in possesso dell'attaccante. Un modo per realizzarlo è necessario per un attaccante copiare un token di una sessione e usarlo in un messaggio http per una sessione diversa, ciò è facile quando il token p disponibile al browser, questo è noto come attacco "cut and paste".

L'Implicit Flow of OAuth 2.0 [RFC6749] non è progettato per attenuare questo rischio. Nella sezione 10.16, si richiede normativamente che qualsiasi utilizzo del processo di autorizzazione come una form di delega di autorizzazione dell'Utente Finale al Client non debba utilizzare l'Implicit Flow senza impiegare i meccanismi di sicurezza aggiuntivi che consentono al Client di determinare se l'ID Token e l'Access Token sono stati emessi per il suo uso.

Nell'OpenID Connect, questo è mitigato attraverso i meccanismi forniti dall'ID Token. L'ID Token è un token di sicurezza firmato che fornisce dichiarazione quali ISS (emittente), SUB (soggetto), AUD (audience), AZP (parti autorizzate), at_hash (hash dell'access token) e c_hash (hash del code). Utilizzando l'ID Token, il Client è in grado di rilevare l'attacco di Topen Substitution.

Il *c_hash* dell'ID Token consente ai Client di evitare la sostituzione del Authorization Code. Il *at_hash del*'ID Token consente ai Client di evitare la sostituzione dell'Access Token.

Inoltre, un utente malintenzionato può tentare di rappresentare un utente con più privilegi sovvertendo il canale di comunicazione tra l'Authorization Endpoint ed il Client, o tra il Token Endpoint ed il Client, ad esempio scambiando l'Authorization Code o riordinando i messaggi, per convincere il Token Endpoint che la concessione di autorizzazione dell'attaccante corrisponde ad un grant inviato per conto di un utente più privilegiato.

Per il binding HTTP definito da questa specifica, le risposte alle richieste di token sono legate alle corrispondenti richieste nel messaggio HTTP, in quanto sia la risposta contenente il token e le richieste sono protette da TLS, il quale può rilevare e prevenire il riordino dei pacchetti.

Quando si progetta un altro binding di questa specifica per un protocollo incapace di richieste e risposte robuste del Token endpoint, questo problema deve essere utilizzato per affrontare ulteriori meccanismi. Tale meccanismo potrebbe essere quello di includere un ID Token con un reclamo c_hash nella richiesta e nella risposta di token.

Timing Attack

Un Timing Attack consente al malintenzionato di ottenere una grande quantità superflua di informazioni attraverso le differenze di tempo trascorso nei percorsi di codice prese dalle operazioni di decrittazione o dalla

convalida della firma di un messaggio. Ciò può essere usato per ridurre la lunghezza effettiva della chiave del cifrario utilizzato.

Le implementazioni non devono terminare il processo di convalida al momento della constatazione di un errore, ma dovrebbero continuare a funzionare fino a quando tutti gli ottetti sono stati elaborati al fine di evitare questo tipo di attacco.

Crypto Related Attacks

Ci sono vari possibili attacchi relativi al crypto a seconda del metodo utilizzato per la crittografia e la firma/controllo dell'integrità. Gli Implementatori devono consultare le Considerazioni di sicurezza per la specifica JWT [JWT] e le specifiche a cui fa riferimento, per evitare le vulnerabilità individuate nel presente capitolo.

Signing and Encryption Order

Le Firme sul testo cifrato non sono considerate valide in molte giurisdizioni. Pertanto, per l'integrità e il non ripudio, questa specifica richiede che la firma del testo normale JSON Claims, quando viene eseguita la firma. Se sono richieste sia la firma che la crittografia, viene eseguita prima la firma JWS e dopo la crittografia JWE.

Issuer Identifier

L'OpenID Connect supporta più emittenti per ogni Host e combinazione di Port. L'emittente restituito dalla ricerca deve corrispondere esattamente al valore della iss in ID Token.

L'OpenID Connect tratta il componente di tracciato di qualsiasi emittente URI come parte dell'Issuer Identifier. Ad esempio, il soggetto "1234" con un Issuer Identifier di "https://example.com" non è equivalente al tema "1234" con Issuer Identifier di "https://example.com/sales".

Si raccomanda di utilizzare una sola emittente per host. Tuttavia, se un host supporta più client, possono essere necessari più emittenti per tale host.

Implicit Flow Threats

Nel Implicit Flow, l'Access Token viene restituito nella componente frammento della redirect_uri del Cliente tramite HTTPS, quindi è protetta tra OP e l'User Agent, e tra l'User Agent e la RP. L'unico posto dove può essere

catturato è l'User Agent in cui la sessione TLS è terminata, il che è possibile se l'User Agent è stato infettato da malware o sotto il controllo di un malintenzionato.

TLS Requirements

Le Implementazioni devono sostenere il protocollo TLS.
Per la protezione contro la divulgazione di informazioni e la manomissione, la protezione della riservatezza deve essere applicato utilizzando TLS con una ciphersuite che garantisca la riservatezza e l'integrità di protezione.
Ogni volta che viene utilizzato TLS, deve essere controllato un certificato del server TLS, per RFC 6125 [RFC6125].

Lifetimes of Access Tokens and Refresh Tokens

L'Access Token potrebbero non essere revocabile dall'Authorization Server. La vita di un Access Token dovrebbe pertanto essere tenuta per un uso singolo o periodi di vita molto brevi.
Se è necessario un accesso continuo all'UserInfo Endpoint o alle altre risorse protette, può essere utilizzato un Refresh token. Il Client può quindi scambiare il Refresh Token al token Endpoint per un nuova breve durata dell'Access token e può essere utilizzato per accedere alla risorsa.
L'Authorization Server deve identificare chiaramente le autorizzazioni a lungo termine per l'utente durante l'autorizzazione. Inoltre, dovrebbe fornire un meccanismo per l'utente finale per revocare l'Accesso Tokens e il Refresh Token concessi ad un Client.

Symmetric Key Entropy

Nella Sezione 10.1 e nella Sezione 10.2, le chiavi sono derivate dal valore *client_secret*. Così, quando si utilizzato le operazioni di firma o crittografia simmetriche, i valori *client_secret* devono contenere l'entropia sufficiente per generare chiavi crittograficamente forti. Inoltre, i valori *client_secret* devono anche contenere almeno il minimo del numero di ottetti necessari per le chiavi MAC per il particolare algoritmo utilizzato. Così, per esempio, per HS256, il valore *client_secret* deve contenere almeno 32 ottetti (e quasi certamente dovrebbe contenerne di più, poiché i valori *client_secret* sono propensi ad usare un alfabeto ristretto).

Need for Signed Requests

In alcune situazioni, i Client potrebbero avere bisogno di utilizzare le richieste firmate per garantire che i parametri della richiesta desiderati siano

consegnati all'OP senza essere state manomesse. Ad esempio, il max_age e acr_values forniscono maggiori garanzie circa la natura della autenticazione eseguita al momento della consegna delle richieste firmate.

Need for Encrypted Requests

In alcune situazioni, conoscere il contenuto di una richiesta di OpenID Connect può rivelare informazioni sensibili circa l'utente finale. Ad esempio, sapendo che il Client richiede un particolare Claim o che richiede che un particolare metodo di autenticazione da utilizzare può rivelare informazioni sensibili sulla utente finale. OpenID Connect consente la crittografia delle richieste all'OpenID provider per evitare che tali informazioni potenzialmente sensibili possa essere rivelato.

Personally Identifiable Information

La risposta dell'UserInfo contiene tipicamente Personally Identifiable Information (PII). Come tale, il consenso dell'utente finale per il rilascio delle informazioni per lo scopo specificato deve essere ottenuto in corrispondenza o prima dell'orario di autorizzazione in conformità alla normativa in materia. Lo scopo di utilizzo è in genere registrato in collaborazione con il redirect_uris.
Solo i dati UserInfo necessari devono essere conservati presso il Client e il client dovrebbe associare i dati ricevuti con la finalità di utilizzo dichiarazione.

Data Access Monitoring

Il Resource Server dovrebbe rendere fruibile i log di accesso alle UserInfo agli utenti finali, in modo tale che possano controllare chi accede ai propri dati.

Correlation

Per proteggere l'utente finale da una possibile correlazione tra i client, deve essere considerato l'uso di un Pairwise Pseudonymous Identifier (PPID) come valore sub (subject).

Offline Access

L'accesso offline consente l'accesso ai Claims quando l'utente non è presente, ciò rappresenta un maggiore rischio alla privacy rispetto al trasferimento di claims quando l'utente è presente. Pertanto, è prudente ottenere il consenso esplicito per l'accesso offline alle risorse. Questa specifica impone l'utilizzo

del parametro prompt per ottenere il consenso a meno che non sia già noto che la domanda soddisfa le condizioni per il trattamento della richiesta in ogni giurisdizione.
Quando un acsess token viene restituito tramite User Agent utilizzando l'Implicit Flow o l'Hybrid Flow, vi è un maggiore rischio di essere esposto ad un attaccante, che in seguito potrebbe utilizzarle per accedere all'UserInfo endpoint. Se l'Acees token non consente l'accesso offline ed il server può distinguere se la richiesta del client è stata fatta offline o online, il rischio sarà notevolmente ridotto. Si noti che può essere difficile distinguere la differenza tra l'accesso online e offline soprattutto per i client nativi. Inoltre, il rischio di esposizione per l'Access Token consegnato attraverso la User Agent per i tipi di risposta di token di codice e token è lo stesso. Così, le implementazioni devono essere pronte a rilevare se l'Access il token è stato rilasciato attraverso l'User Agent o direttamente dal Token Endpoint e negare l'accesso offline se il token è stato rilasciato attraverso l'User Agent.

3 Analisi di Sicurezza

Nel capitolo precedente è stata presentata una descrizione informale dell'OpenID Connect basata sulla documentazione ufficiale fornita dall'OpenID Foundation e completa degli esempi presenti sulla stessa.

In questo capitolo sarà illustrato in dettaglio lo studio formale del protocollo suddetto. In particolare nella sezione 1 sarà riportata la formalizzazione del protocollo in notazione Alice-Bob e, nella sezione 2, sarà fornita la specifica HLPSL corrispondente e la simulazione effettuata con il tools SPAN. Infine, saranno presentati i risultati ottenuti dall'analisi automatica di diverse proprietà di sicurezza del protocollo mediante i back-end di AVISPA ed alcune considerazioni di sicurezza.

3.1 Notazione Alice-Bob

Come detto in precedenza, questo lavoro riguarda l'analisi dell'**OpenID Connect Core 1.0**, ed in particolare il metodo d'autenticazione denominato: **Authorization Code Flow**.

Il protocollo OpenID Connect Core può essere genericamente schematizzato i due fasi principali:

1. Autenticazione dell'Utente Finale per conto del Client
2. Acquisizione delle informazioni di base sul profilo dell'Utente Finale (Claims)

Gli attori coinvolti nel processo sono:

- RP = Relying Party: l'applicazione Client che usa l'OpenID Connect
- OP = OpenID Provider: il Server di Autenticazione che implementa l'OpenID Connect
- EU = End-User: l'Utente finale
- TE = Token EndPoint: il Server che rilascia l'ID Token e l'Access Token
- UE = UserInfo EndPoint: il Server che rilascia i Claims

Di seguito si riporta la specifica astratta per l'autenticazione ed il rilascio dell'ID Token:

1. RP -> OP: {Authentication_Request (response_type, scope, client_id, state, redirect_uri}_ROKey
2. OP -> EU: End-User_Authentication (*)
3. EU -> OP: End-User_Consent/Authorization (*)
4. OP -> RP: {Authentication_Response (code, state)}_ROKey
5. RP -> TE: {Token_Request (grant_type, code, redirect_uri)}_RTKey
6. TE -> RP: {Token_Response (access_token, token_type, refresh_token, expires_in, {id_token}_CSKey)}_RTKey

Mentre la specifica astratta che riguarda il rilascio di ulteriori Claims è la seguente:

7. RP -> UE: {UserInfo_Request (access token)}_RUKey (**)
8. UE -> RP: {UserInfo_Response (claims)}_RUKey (**)

(*) Non implementata perché le specifiche esulano dal protocollo OpenID Connect
(**) Non implementata in quanto le caratteristiche di specifica sono identiche ai passi 5 e 6

3.1.1 Scelte implementative

Le scelte implementative effettuate richiedono alcune osservazioni.
In primo luogo, non essendo state implementate le funzionalità OpenID Connect Discovery e OpenID Connect Dynamic Registration, si presuppone che il Client (Relying Party) conosca l'indirizzo dell'Authorization Server (OpenID Provider) e sia già registrato sullo stesso e, di conseguenza, sia in possesso di un *client_id* valido (codice identificativo rilasciato al momento della registrazione del Client presso l' Authorization Server) e del relativo *client_secret* (segreto condiviso tra il Client e l' Authorization Server).

Inoltre, le sessioni sicure TLS sono state modellizzate supponendo le comunicazioni cifrate con una chiave simmetrica condivisa dagli attori:

- ROKey è condivisa tra RP (Relying Party) e OP (OpenID Provider)
- RTKey è condivisa tra RP (Relying Party) e TE (Token EndPoint)
- RUKey è condivisa tra RP (Relying Party) e UE (User EndPoint)
- CSKey, calcolata dal client_secret, è utilizzata per firmare/cifrare il contenuto dei messaggi.

In sede di formalizzazione si è deciso che l'utente non sia collegato al momento dell'inizio della sessione. Pertanto, la specifica è stata semplificata escludendo lo scambio di messaggi tra l'OpenID Provider e l'End-User (passi 2 e 3), in quanto estranei al protocollo OpenID Connect.

Infine, l'ID Token deve essere firmato, per esempio utilizzando la *CSKey,* dall'Emettitore (in questo caso dal Token Endpoint). Se l'ID Token viene ricevuto tramite la comunicazione diretta tra il Client ed il Token Endpoint (come in questo flusso), la convalida del server TLS può essere utilizzata per nvalidare l'emittente al posto di controllare la firma del token.

3.1.2 Messaggi scambiati

Di seguito sono riportati i messaggi d'esempio, come riportati sulla specifica, con i quali è stata sviluppata l'analisi delle proprietà di sicurezza.

Authentication Request:

```
response_type=code
      // Identifica il flusso: Authorization Code Flow
scope=openid profile email
      // Scopo della richiesta OpenID Connect
client_id=s6BhdRkqt3
      // Identifica ClientId valido per Authorization Server
state=af0ifjsldkj
      // Valore per lo stato dalla richiesta alla risposta
redirect_uri=https//client.example.org
      // l'URI a cui verrà inviata la risposta
```

Successful Authentication Response:

```
code=SplxlOBeZQQYbYS6WxSbIA
      // Identifica l'Authorization Code
state=af0ifjsldkj
      // Valore per lo stato dalla richiesta alla risposta
```

Authentication Error Response:

```
error=invalid_request
      // Codice dell'errore
error_description= Unsupported response_type value
      // Descrizione errore
state=af0ifjsldkj
      // Valore per lo stato dalla richiesta alla risposta
```

Token Request:

```
grant_type=authorization_code
      // Identifica il tipo di Grant
code=SplxlOBeZQQYbYS6WxSbIA
      // Identifica l'Authorization Code
redirect_uri=https://client.example.org
      // l'URI per la risposta
```

Successful Token Response:

```
{
 "access_token": "SlAV32hkKG",
 "token_type": "Bearer",
```

```
   "refresh_token": "8xLOxBtZp8",
   "expires_in": 3600,
   "id_token":
"eyJhbGciOiJSUzI1NiIsImtpZCI6IjFlOWdkazcifQ.ewogImlzc
yI6ICJodHRwOi8vc2VydmVyLmV4YW1wbGUuY29tIiwKICJzdWIiOiAiMjQ4Mjg5
NzYxMDAxIiwKICJhdWQiOiAiczZCaGRSa3F0MyIsCiAibm9uY2UiOiAibi0wUzZ
fV3pBMk1qIiwKICJleHAiOiAxMzExMjgxOTcwLAogImlhdCI6IDEzMTEyODA5Nz
AKfQ.ggW8hZ1EuVLuxNuuIJKX_V8a_OMXzR0EHR9R6jgdqrOOF4daGU96Sr_P6q
Jp6IcmD3HP99Obi1PRs-cwh3LO-p146waJ8IhehcwL7F09JdijmBqkvPeB2T9CJ
NqeGpe-gccMg4vfKjkM8FcGvnzZUN4_KSP0aAp1tOJ1zZwgjxqGByKHiOtX7Tpd
QyHE5lcMiKPXfEIQILVq0pc_E2DzL7emopWoaoZTF_m0_N0YzFC6g6EJbOEoRoS
K5hoDalrcvRYLSrQAZZKflyuVCyixEoV9GfNQC3_osjzw2PAithfubEEBLuVVk4
XUVrWOLrLl0nx7RkKU8NXNHq-rvKMzqg"  // Signed dal Token Endpoint
      }
```

Token Error Respons:

```
{
   "error": "invalid_request"      // Token Error Response
  }
```

Set of Claims contenuti nell'ID Token:

```
ID token          // Contenuto del ID Token
{
   "iss": "https://server.example.com",
   "sub": "24400320",
   "aud": "s6BhdRkqt3",
   "exp": 1311281970,
   "iat": 1311280970,
   "auth_time": 1311280969,
   "acr": "urn:mace:incommon:iap:silver"
  }
```

UserInfo Request:

```
Authorization: Bearer SlAV32hkKG   // Access Token
```

Successful UserInfo Response:

```
{
   "sub": "248289761001",
   "name": "Jane Doe",
   "given_name": "Jane",
   "family_name": "Doe",
   "preferred_username": "j.doe",
   "email": "janedoe@example.com",
   "picture": "http://example.com/janedoe/me.jpg"
  }
```

UserInfo Error Response:

```
error="invalid_token",
      // Codice dell'errore
error_description="The Access Token expired"
      // Descrizione errore
```

3.2 Specifica HLPSL

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                  %
%             OpendID Connect Core 1.0             %
%                                                  %
%               February 25, 2014                  %
%                                                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                  %
% E' assunto che il Client (Relying Party) sia già %
%    registrato sull'Authorization Server (OpendID %
%    Provider) tramite [Client_ID],[Client_Secret] %
%                                                  %
% [CSKey] rappresenta la chiave simmetrica segreta %
%    derivata dal valore Client_Secret e condivisa %
%    tra Client (RP) ed Authorization Server (OP)  %
%                                                  %
% Le comunicazioni tra Agenti utilizzano il TLS    %
%     si assume cha la chiave simmetrica per la    %
%     sessione di comunicazione è così definita:   %
%     [ROKey] Relying Party <=> OpenID Provider    %
%     [RTKey] Relying Party <=> Token End-Point    %
%     [RUKey] Relying Party <=> UserInfoEnd-Point  %
%                                                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


role relyingparty      (
      RP,OP,TE          :     agent,
      ClientId, URI     :     text,
      ROKey,RTKey,CSKey :     symmetric_key,
      SND,RCV           :     channel(dy) )
played_by RP
def=
      local
            State       :     nat,
            Stat,Code,IdToken : text
      init
            State := 0
      transition

      1.    State=0 /\ RCV(start) =|>

%% 1A. SND Authentication Request

                  State':=1
                  /\ Stat':=new()
                  /\ SND({ClientId.Stat'.URI}_ROKey)
                  /\ secret(Stat',sec_1,{RP,OP})
                  /\ witness(RP,OP,auth_1,Stat')
```

```
%% 4B. RCV Authentication Response

      2.    State=1 /\ RCV({Code'.Stat'}_ROKey) =|>

%% 5A. SND Token Request

                  State':=2
                  /\ SND({Code'.URI}_RTKey)
                  /\ secret(Stat',sec_1,{RP,OP})
                  /\ secret(Code',sec_2,{RP,OP})
                  /\ witness(RP,TE,auth_3,Code')

%% 6B. RCV Token Response

      3.    State=2 /\ RCV({{IdToken'}_CSKey}_RTKey) =|>
                  State':=3
                  /\ secret(IdToken',sec_3,{RP,TE})
end role

role openidprovider     (
      OP,RP,TE          :     agent,
      ClientId, URI     :     text,
      ROKey             :     symmetric_key,
      SND,RCV           :     channel(dy) )
played_by OP
def=
      local
            State       :     nat,
            Code,Stat   :     text
      init
            State := 0
      transition

%% 1B. RCV Authentication Request

      1.    State=0 /\ RCV({ClientId.Stat'.URI}_ROKey) =|>

%% 4A. SND Authentication Response
                  State':=1
                  /\ Code':=new()
                  /\ SND({Code'.Stat'}_ROKey)
                  /\ secret(Stat',sec_1,{RP,OP})
                  /\ secret(Code',sec_2,{RP,OP})
                  /\ wrequest(OP,RP,auth_1,Stat')
end role

role tokenendpoint      (
      TE,RP,OP          :     agent,
      URI               :     text,
      RTKey,CSKey       :     symmetric_key,
      SND,RCV           :     channel(dy) )
played_by TE
def=
      local
```

```
            State       :     nat,
            Code,IdToken:     text
      init
            State := 0
      transition
```

%% 5B. RCV Token Request

```
      1.    State=0 /\ RCV({Code'.URI}_RTKey) =|>
```

%% 6A. SND Token Response

```
                  State':=1
                  /\ IdToken':=new()
                  /\ SND({{IdToken'}_CSKey}_RTKey)
                  /\ secret(Code',sec_2,{RP,OP})
                  /\ secret(IdToken',sec_3,{RP,TE})
                  /\ wrequest(TE,RP,auth_3,Code')
end role


role session      (
      RP,OP,TE          :     agent,
      ClientId,URI      :     text,
      ROKey,RTKey,CSKey :     symmetric_key )
def=
      local
            SND1,RCV1,SND2,RCV2,SND3,RCV3 :     channel(dy)
      composition
            relyingparty(RP,OP,TE,ClientId,URI,ROKey,RTKey,CSKey,SND1,RCV1)
           /\ openidprovider(OP,RP,TE,ClientId,URI,ROKey,SND2,RCV2)
           /\ tokenendpoint(TE,RP,OP,URI,RTKey,CSKey, SND3,RCV3)
end role

role environment()
def=
      const
            rp,op,te    :     agent,
            clientid,uri:     text,
            rokey,rtkey,rikey,cskey:symmetric_key,
            auth_1,auth_2,auth_3,
            sec_1,sec_2,sec_3: protocol_id
      intruder_knowledge = {rp,op,te,rikey,i}
      composition
```

%% Senza Intruder

```
      session(rp,op,te,clientid,uri,rokey,rtkey,cskey)
      /\ session(rp,op,te,clientid,uri,rokey,rtkey,cskey)
```

%% L'intruder è il Client: Relying Party

```
      /\ session(i,op,te,clientid,uri,rikey,rikey,cskey)
```

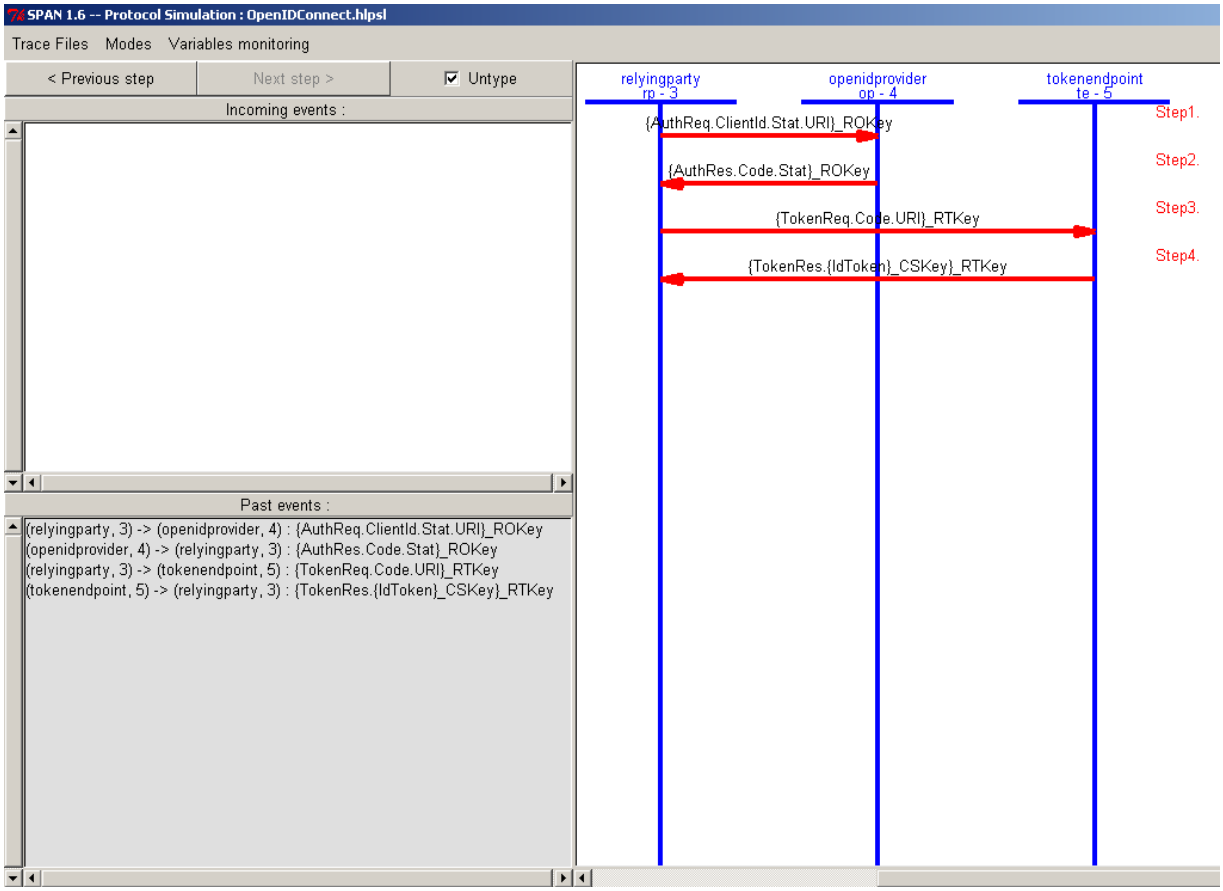
```
%% L'intruder è l'Authentication Server: OpenID Provider

      /\ session(rp,i,te,clientid,uri,rikey,rtkey,cskey)

%% L'intruder è il Token End-Point

      /\ session(rp,op,i,clientid,uri,rokey,rikey,cskey)

end role

goal

      authentication_on auth_1      %% Autentica RP su OP con Stat
      authentication_on auth_3      %% Autentica RP su TE con Code
      secrecy_of sec_1              %% Stat segreto tra RP e OP
      secrecy_of sec_2              %% Code segreto tra RP e OP
      secrecy_of sec_3              %% IdToken segreto tra RP e TE

end goal

environment()
```

3.2.1 Simulazione

Di seguito è riportata la simulazione dell'esecuzione del protocollo realizzata con il tool SPAN (Security Protocol ANimator for AVISPA).

SPAN costruisce interattivamente, a partire dalla specifica HLPSL di un protocollo, il diagramma di sequenza (Message Sequence Chart, MSC) della sua esecuzione.

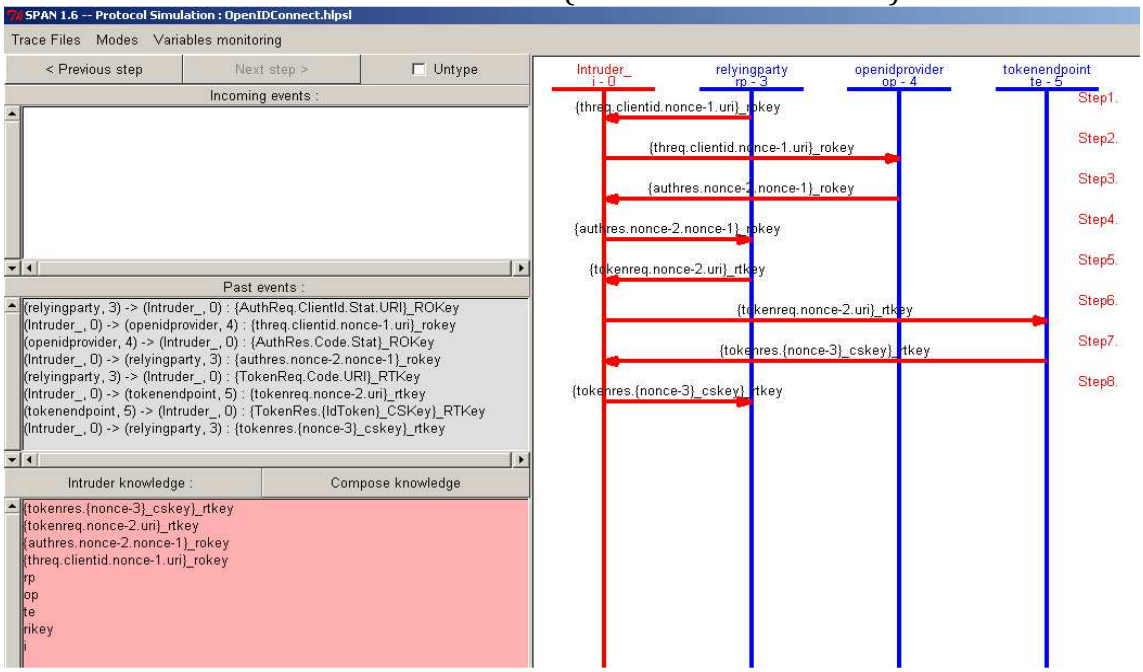
Di seguito è visibile l'MSC relativo alla Sessione 1 della specifica HLPSL, ovvero quella senza *Intruder*:



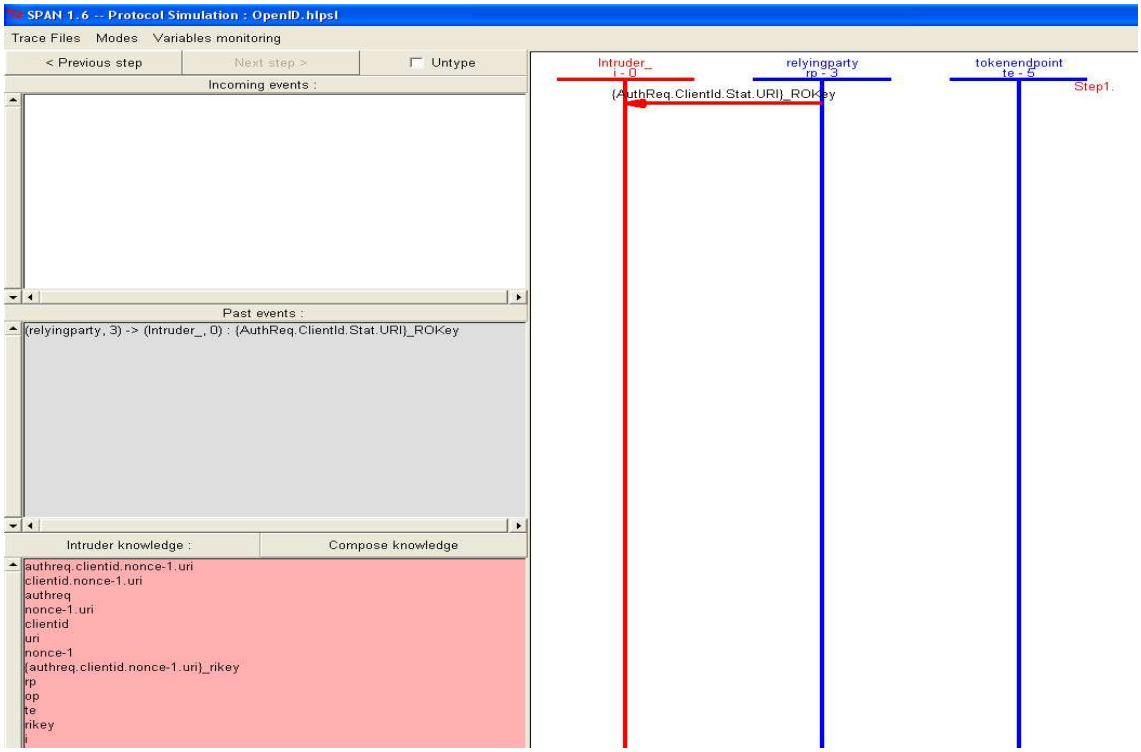
Ogni colonna del grafico rappresenta un Ruolo (*Entità*) del protocollo.

Il tool evidenzia, inoltre, la sequenza dei passi del protocollo, etichettandoli con *Step1,….,StepN*, ed i messaggi scambiati, con i relativi valori e verso.

Della stessa sessione si riporta il grafico risultante dalla simulazione con l'Iintruder che riveste il Ruolo della Rete (*Intruder is the Network*):

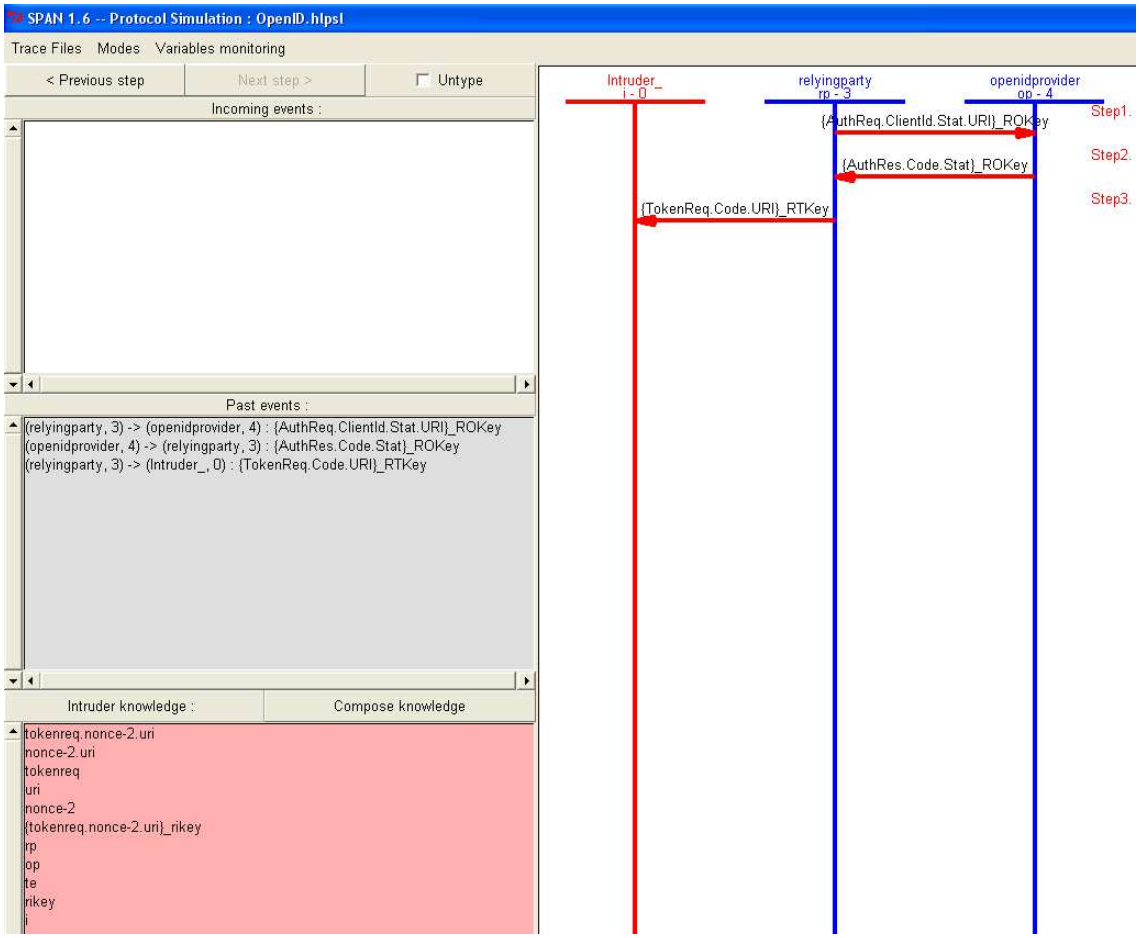


La sessione 3, riferita alla simulazione in cui l'Intruder riveste il Ruolo dell'Authorization Server (*OpenID Provider*), si evince immediatamente una violazione relativa alla confidenzialità:



Come è possibile rilevare dal grafico, la simulazione si interrompe allo *Step1* perché l'Intruder (*OpenId Provider*) riceve in chiaro il contenuto su cui era stato imposto un Goal of Secrecy. In particolare il valore rappresentato dal campo *Stat,* che solitamente viene utilizzato per identificare lo stato di sessione tra il Client (*Relying Party*) e l'Authorization Server (*OpenId Provider*) durante tutte le fasi del protocollo.

Con la simulazione riferita alla Sessione 4 della specifica HLPSL, quella in cui l'Intruder riveste il Ruolo *dell'Token EndPoint,* si registra un'altra interruzione.



In quest'altro caso la simulazione si interrompe allo *Step 3* perché l'Intruder (*Token EndPoint*) riceve in chiaro il contenuto su cui era stato imposto un Goal of Secrecy. In particolare il valore contenuto nel campo *Code* che rappresenta la Risposta Positiva di Autorizzazione rilasciata dall'Authorization Server (*OpenId Provider*) al Client (*Relying Party*).

3.2.2 Risultati di AVISPA

3.2.2.1 Risultato del Back-End: **OFMC**

```
% OFMC
% Version of 2006/02/13
SUMMARY
  UNSAFE
DETAILS
  ATTACK_FOUND
PROTOCOL
  /home/avispa/web-interface-
computation/./tempdir/workfileM6Jx0a.if
GOAL
  secrecy_of_sec_2
BACKEND
  OFMC
COMMENTS
STATISTICS
  parseTime: 0.00s
  searchTime: 0.21s
  visitedNodes: 17 nodes
  depth: 2 plies
ATTACK TRACE
i -> (rp,15): start
(rp,15) -> i: {clientid.Stat(1).uri}_rikey
i -> (rp,15): {x267.x255}_rikey
(rp,15) -> i: {x267.uri}_rtkey
i -> (te,3): {x267.uri}_rtkey
(te,3) -> i: {{IdToken(3)}_cskey}_rtkey
i -> (i,17): x267
i -> (i,17): x267


% Reached State:
%
% secret(x267,sec_2,set_118)
% secret(IdToken(3),sec_3,set_119)
% contains(rp,set_118)
% contains(op,set_118)
% contains(rp,set_119)
% contains(te,set_119)
% secret(x255,sec_1,set_146)
% secret(x267,sec_2,set_147)
% witness(rp,te,auth_3,x267)
% contains(rp,set_146)
% contains(i,set_146)
% contains(rp,set_147)
% contains(i,set_147)
% secret(Stat(1),sec_1,set_145)
% witness(rp,i,auth_1,Stat(1))
```

```
% contains(rp,set_145)
% contains(i,set_145)
%
state_relyingparty(rp,i,te,clientid,uri,rikey,rtkey,cskey,2,x255,x
267,dummy_nonce,set_145,set_146,set_147,set_148,15)
%
state_tokenendpoint(te,i,op,uri,rikey,cskey,0,dummy_nonce,dummy_no
nce,set_142,set_143,12)
%
state_openidprovider(op,i,te,clientid,uri,rikey,0,dummy_nonce,dumm
y_nonce,set_139,set_140,12)
%
state_tokenendpoint(te,rp,op,uri,rtkey,cskey,0,dummy_nonce,dummy_n
once,set_132,set_133,7)
%
state_relyingparty(rp,op,te,clientid,uri,rokey,rtkey,cskey,0,dummy
_nonce,dummy_nonce,dummy_nonce,set_124,set_125,set_126,set_127,7)
%
state_openidprovider(op,rp,te,clientid,uri,rokey,0,dummy_nonce,dum
my_nonce,set_129,set_130,7)
%
state_tokenendpoint(te,rp,op,uri,rtkey,cskey,1,x267,IdToken(3),set
_118,set_119,3)
%
state_relyingparty(rp,op,te,clientid,uri,rokey,rtkey,cskey,0,dummy
_nonce,dummy_nonce,dummy_nonce,set_97,set_98,set_99,set_100,3)
%
state_openidprovider(op,rp,te,clientid,uri,rokey,0,dummy_nonce,dum
my_nonce,set_112,set_113,3)
%
state_relyingparty(rp,op,i,clientid,uri,rokey,rikey,cskey,0,dummy_
nonce,dummy_nonce,dummy_nonce,set_155,set_156,set_157,set_158,19)
%
state_openidprovider(op,rp,i,clientid,uri,rokey,0,dummy_nonce,dumm
y_nonce,set_160,set_161,19)
%
state_tokenendpoint(te,rp,i,uri,rtkey,cskey,0,dummy_nonce,dummy_no
nce,set_152,set_153,17
```

Il Back-End OFMC rileva la vulnerabilità del Goal of Secrecy imposto nella specifica HLPSL sul campo *Code,* come già rivelato nella simulazione precedente.

3.2.2.2 Risultato del Back-End: **Cl-AtSe**

```
SUMMARY
  UNSAFE

DETAILS
```

```
  ATTACK_FOUND
  TYPED_MODEL

PROTOCOL
  /home/avispa/web-interface-
computation/./tempdir/workfileM6Jx0a.if

GOAL
  Secrecy attack on (n7(Code))

BACKEND
  CL-AtSe

STATISTICS

  Analysed   : 178 states
  Reachable  : 24 states
  Translation: 0.06 seconds
  Computation: 0.00 seconds


ATTACK TRACE
 i -> (rp,15):  start
 (rp,15) -> i:  {clientid.n25(Stat).uri}_rikey
                & Secret(n25(Stat),set_145);  Add rp to set_145;
Add i to set_145;

 i -> (rp,19):  start
 (rp,19) -> i:  {clientid.n33(Stat).uri}_rokey
                & Secret(n33(Stat),set_155);
Witness(rp,op,auth_1,n33(Stat));  Add rp to set_155;  Add op to
set_155;

 i -> (op,4):  {clientid.n33(Stat).uri}_rokey
 (op,4) -> i:  {n7(Code).n33(Stat)}_rokey
               & Secret(n7(Code),set_113);
Secret(n33(Stat),set_112);  Add rp to set_112;  Add op to set_112;
Add rp to set_113;  Add op to set_113;

 i -> (rp,19):  {n7(Code).n33(Stat)}_rokey
 (rp,19) -> i:  {n7(Code).uri}_rikey
                & Secret(n7(Code),set_157);
Secret(n33(Stat),set_156);  Add rp to set_156;  Add op to set_156;
Add rp to set_157;  Add op to set_157;
```

Anche il Back-End Cl-AtSe rileva la vulnerabilità del Goal of Secrecy imposto nella specifica HLPSL sul campo *Code*.

3.2.2.3 Risultato del Back-End: **SATMC**

```
SUMMARY
  UNSAFE

DETAILS
  ATTACK_FOUND
  STRONGLY_TYPED_MODEL
  BOUNDED_NUMBER_OF_SESSIONS
  BOUNDED_SEARCH_DEPTH
  BOUNDED_MESSAGE_DEPTH

PROTOCOL
  workfileM6Jx0a.if

GOAL
  secrecy_of_sec_2(code(op,4),set_113)
  secrecy_of_sec_2(code(op,4),set_157)
  secrecy_of_sec_2(code(op,4),set_99)

BACKEND
  SATMC

COMMENTS

STATISTICS
  attackFound               true      boolean
  upperBoundReached         false     boolean
  graphLeveledOff           no        boolean
  satSolver                 zchaff    solver
  maxStepsNumber            11        steps
  stepsNumber               4         steps
  atomsNumber               1330      atoms
  clausesNumber             4349      clauses
  encodingTime              0.93      seconds
  solvingTime               0.0       seconds
  if2sateCompilationTime    0.2       seconds

ATTACK TRACE
  i         ->    (rp,15)   : start
  (rp,15)   ->    i         : {clientid.stat(rp,15).uri}_rikey
  i         ->    (rp,19)   : start
  (rp,19)   ->    i         : {clientid.stat(rp,19).uri}_rokey
  i         ->    (rp,7)    : start
  (rp,7)    ->    i         : {clientid.stat(rp,7).uri}_rokey
  i         ->    (rp,3)    : start
  (rp,3)    ->    i         : {clientid.stat(rp,3).uri}_rokey
  i         ->    (op,12)   : {clientid.stat(rp,15).uri}_rikey
  (op,12)   ->    i         : {code(op,12).stat(rp,15)}_rikey
  i         ->    (op,4)    : {clientid.stat(rp,3).uri}_rokey
  (op,4)    ->    i         : {code(op,4).stat(rp,3)}_rokey
  i         ->    (te,13)   : {clientid.uri}_rikey
  (te,13)   ->    i         : {{idtoken(te,13)}_cskey}_rikey
```

```
  i        ->    (rp,19)   : {code(op,4).stat(rp,3)}_rokey
  (rp,19)  ->    i         : {code(op,4).uri}_rikey
  i        ->    (rp,3)    : {code(op,4).stat(rp,3)}_rokey
  (rp,3)   ->    i         : {code(op,4).uri}_rtkey
  i        ->    (te,17)   : {code(op,4).uri}_rtkey
  (te,17)  ->    i         : {{idtoken(te,17)}_cskey}_rtkey
  i        ->    (rp,19)   : {{idtoken(te,13)}_cskey}_rikey
  i        ->    (rp,15)   : {code(op,12).stat(rp,15)}_rikey
  (rp,15)  ->    i         : {code(op,12).uri}_rtkey
```

Lo stesso esito lo fornisce il Back-End SATMC.

3.2.2.4 Risultato del Back-End: **TA4SP**

```
SUMMARY
   INCONCLUSIVE

DETAILS
   TIME_OUT

PROTOCOL
   /home/avispa/web-interface-
computation/./tempdir/workfileM6Jx0a.if

GOAL
   SECRECY

BACKEND
   TA4SP

COMMENTS
   Computation broke
```

L'ultimo Back-End TA4SP si interrompe in uno stato *Inconclusive* e, pertanto, non fornisce nessun elemento utile all'analisi che si sta effettuando.

Ciò rappresenta uno dei limiti del tool AVISPA.

3.3 Analisi di sicurezza

L'analisi del protocollo OpenID Connect è stata realizzata attraverso l'uso del tools AVISPA direttamente sul sito web: http://www.avispa-project.org/web-interface/expert.php.

Come è possibile rilevare dal codice HLPSL allegato, i Goal ricavati dalla specifica OpenID Connect da sottoporre a verifica formale sono stati individuati nei seguenti:

```
goal
      authentication_on auth_1      %% Autentica RP su OP con ClientID
      authentication_on auth_3      %% Autentica RP su TE con Code
      secrecy_of sec_1              %% Stat segreto tra RP e OP
      secrecy_of sec_2              %% Code segreto tra RP e OP
      secrecy_of sec_3              %% IdToken segreto tra RP e TE
end goal
```

Come è facilmente rilevabile dai risultati dei back-end di AVISPA, è stata riscontrata una vulnerabilità sulla proprietà ***secrecy_of sec_2 [Code tra RP e OP].***

Si rammenta, comunque, che la formalizzazione HLPSL è stata implementata sulla formulazione minimale del protocollo OpenID Connect, ovvero contemplando esclusivamente i valori obbligatori e le caratteristiche di sicurezza di base.

Infatti, leggendo le Considerazioni sulla sicurezza, descritte nel paragrafo 2.6, è possibile individuare soluzioni e contromisure idonee a mitigare un gran numero di vulnerabilità conosciute e che gli strumenti di verifica automatica, come AVISPA, non riescono a prevedere.

In particolare, la vulnerabilità riscontrata sul valore ***Code*** può essere facilmente rimossa attraverso l'utilizzo delle funzioni di firma e crittografia basate sul valore, segreto e condiviso, ***client_secret*** negoziato nella fase di Registrazione del Client (*Relying Party*) sull'Authorization Server (*OpenId Provider*) e che non è stata analizzata nel seguente lavoro. Sul punto si rimanda alla lettura della specifica OpenID Connect Dynamic Client Registration 1.0[26] Le stesse considerazioni effettuate sul campo ***Code*** sono valide per la verifica della proprietà *secrecy* di altri valori, come per esempio ***Stat*** oppure ***IdToken***.

[26] http://openid.net/specs/openid-connect-registration-1_0.html

3.3.1 Specifica HLPSL Corretta

Si riporta di seguito la specifica corretta:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                  %
%            OpendID Connect Core 1.0              %
%                                                  %
%              February 25, 2014 (safe)            %
%                                                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                  %
% E’ assunto che il Client (Relying Party) sia già %
%    registrato sull'Authorization Server (OpendID %
%    Provider) tramite [Client_ID],[Client_Secret] %
%                                                  %
% [CSKey] rappresenta la chiave simmetrica segreta %
%    derivata dal valore Client_Secret e condivisa %
%    tra Client (RP) ed Authorization Server (OP)  %
%                                                  %
% Le comunicazioni tra Agenti utilizzano il TLS    %
%     si assume cha la chiave simmetrica per la    %
%     sessione di comunicazione è così definita:   %
%     [ROKey] Relying Party <=> OpenID Provider    %
%     [RTKey] Relying Party <=> Token End-Point    %
%     [RUKey] Relying Party <=> UserInfoEnd-Point  %
%                                                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


role relyingparty       (
      RP,OP,TE          :     agent,
      ClientId, URI     :     text,
      ROKey,RTKey,CSKey :     symmetric_key,
      SND,RCV           :     channel(dy) )
played_by RP
def=
      local
            State       :     nat,
            Stat,Code,IdToken : text
      init
            State := 0
      transition

      1.    State=0 /\ RCV(start) =|>

%% 1A. SND Authentication Request

                  State':=1
                  /\ Stat':=new()
                  /\ SND({{ClientId.Stat'.URI}_CSKey}_ROKey)
                  /\ secret(Stat',sec_1,{RP,OP})
```

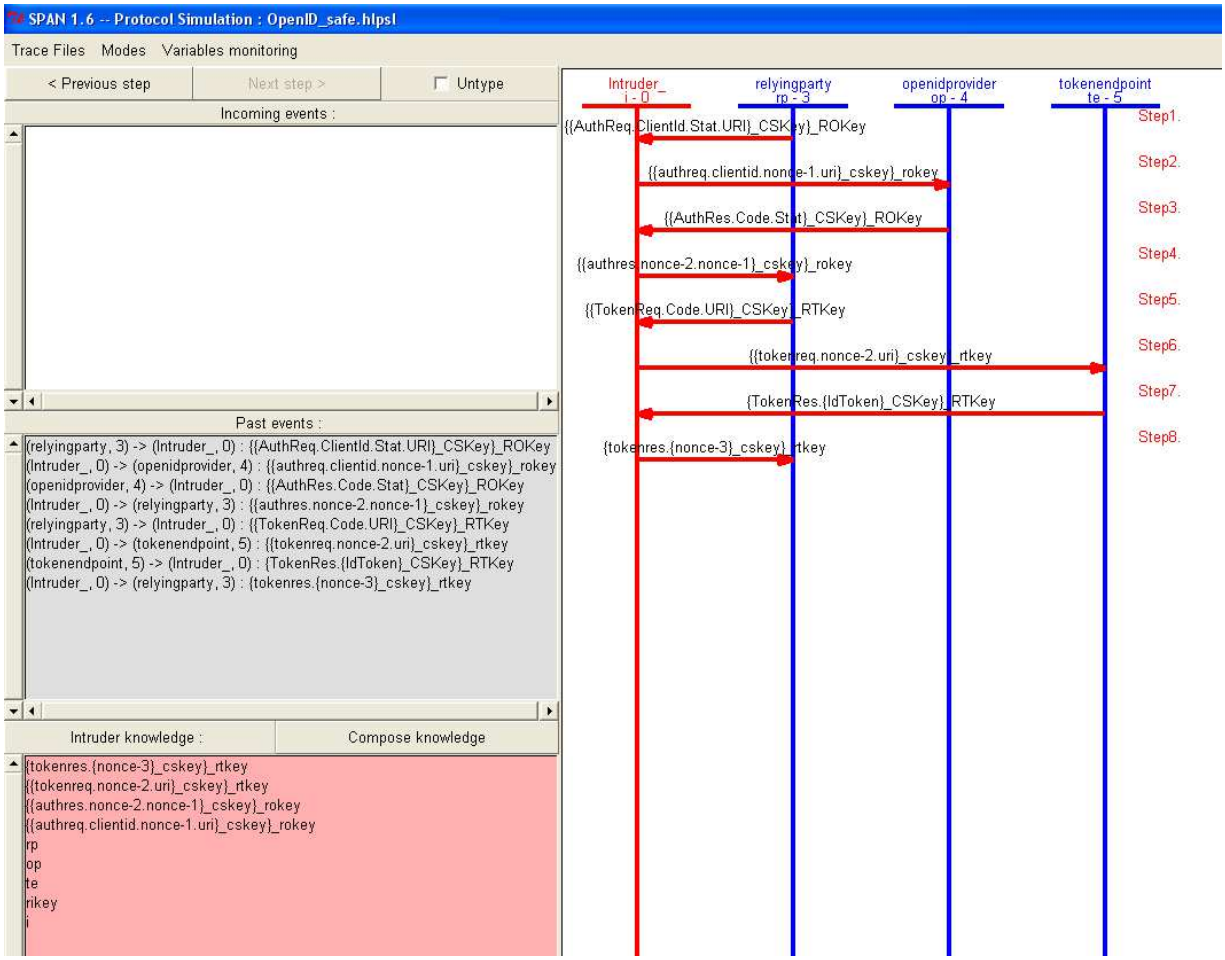
```
                /\ witness(RP,OP,auth_1,Stat')

%% 4B. RCV Authentication Response

      2.    State=1 /\ RCV({{Code'.Stat'}_CSKey}_ROKey) =|>

%% 5A. SND Token Request

                State':=2
                /\ SND({{Code'.URI}_RTKey}_CSKey)
                /\ secret(Stat',sec_1,{RP,OP})
                /\ secret(Code',sec_2,{RP,OP})
                /\ witness(RP,TE,auth_3,Code')

%% 6B. RCV Token Response

      3.    State=2 /\ RCV({{IdToken'}_CSKey}_RTKey) =|>
                State':=3
                /\ secret(IdToken',sec_3,{RP,TE})
end role

role openidprovider     (
      OP,RP,TE          :     agent,
      ClientId, URI     :     text,
      ROKey ,CSKey      :     symmetric_key,
      SND,RCV           :     channel(dy) )
played_by OP
def=
      local
            State       :     nat,
            Code,Stat   :     text
      init
            State := 0
      transition

%% 1B. RCV Authentication Request

      1.    State=0 /\ RCV({{ClientId.Stat'.URI}_CSKey}_ROKey) =|>

%% 4A. SND Authentication Response

                State':=1
                /\ Code':=new()
                /\ SND({{Code'.Stat'}_CSKey}_ROKey)
                /\ secret(Stat',sec_1,{RP,OP})
                /\ secret(Code',sec_2,{RP,OP})
                /\ wrequest(OP,RP,auth_1,Stat')
end role

role tokenendpoint      (
      TE,RP,OP          :     agent,
      URI               :     text,
      RTKey,CSKey :     symmetric_key,
      SND,RCV           :     channel(dy) )
```

```
played_by TE
def=
      local
            State       :     nat,
            Code,IdToken:     text
      init
            State := 0
      transition

%% 5B. RCV Token Request

      1.    State=0 /\ RCV({{Code'.URI}_CSKey}_RTKey) =|>

%% 6A. SND Token Response

                  State':=1
                  /\ IdToken':=new()
                  /\ SND({{IdToken'}_CSKey}_RTKey)
                  /\ secret(Code',sec_2,{RP,OP})
                  /\ secret(IdToken',sec_3,{RP,TE})
                  /\ wrequest(TE,RP,auth_3,Code')
end role


role session      (
      RP,OP,TE          :     agent,
      ClientId,URI      :     text,
      ROKey,RTKey,CSKey :     symmetric_key )
def=
      local
            SND1,RCV1,SND2,RCV2,SND3,RCV3 :     channel(dy)
      composition
            relyingparty(RP,OP,TE,ClientId,URI,ROKey,RTKey,CSKey,SND1,RCV1)
           /\ openidprovider(OP,RP,TE,ClientId,URI,ROKey,CSKey,SND2,RCV2)
           /\ tokenendpoint(TE,RP,OP,URI,RTKey,CSKey,SND3,RCV3)
end role

role environment()
def=
      const
            rp,op,te    :     agent,
            clientid,uri:     text,
            rokey,rtkey,rikey,cskey:symmetric_key,
            auth_1,auth_2,auth_3,
            sec_1,sec_2,sec_3: protocol_id
      intruder_knowledge = {rp,op,te,rikey,i}
      composition

%% Senza Intruder

      session(rp,op,te,clientid,uri,rokey,rtkey,cskey)
      /\ session(rp,op,te,clientid,uri,rokey,rtkey,cskey)

%% L'intruder è il Client: Relying Party
```

```
      /\ session(i,op,te,clientid,uri,rikey,rikey,cskey)

%% L'intruder è l'Authentication Server: OpenID Provider

      /\ session(rp,i,te,clientid,uri,rikey,rtkey,cskey)

%% L'intruder è il Token End-Point

      /\ session(rp,op,i,clientid,uri,rokey,rikey,cskey)

end role

goal

      authentication_on auth_1      %% Autentica RP su OP con Stat
      authentication_on auth_3      %% Autentica RP su TE con Code
      secrecy_of sec_1              %% Stat segreto tra RP e OP
      secrecy_of sec_2              %% Code segreto tra RP e OP
      secrecy_of sec_3              %% IdToken segreto tra RP e TE

end goal

environment()
```

3.3.2 Simulazione Versione Corretta

Si riporta la simulazione effettuata con il tool SPAN con l'Intruder.



3.3.3 Risultati di AVISPA

Risultato del Back-End OFMC:

```
% OFMC
% Version of 2006/02/13
SUMMARY
  SAFE
DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
PROTOCOL
```

```
  /home/avispa/web-interface-
computation/./tempdir/workfileeSZ5g4.if
GOAL
  as_specified
BACKEND
  OFMC
COMMENTS
STATISTICS
  parseTime: 0.00s
  searchTime: 67.60s
  visitedNodes: 8576 nodes
  depth: 12 plies
```

Risultato del Back-End Cl-AtSe:

```
SUMMARY
  SAFE

DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
  TYPED_MODEL

PROTOCOL
  /home/avispa/web-interface-
computation/./tempdir/workfileeSZ5g4.if

GOAL
  As Specified

BACKEND
  CL-AtSe

STATISTICS

  Analysed   : 941093 states
  Reachable  : 146979 states
  Translation: 0.06 seconds
  Computation: 35.16 seconds
```

Risultato del Back-End SATMC:

```
SUMMARY
  SAFE
```

```
DETAILS
  STRONGLY_TYPED_MODEL
  BOUNDED_NUMBER_OF_SESSIONS
  BOUNDED_SEARCH_DEPTH
  BOUNDED_MESSAGE_DEPTH

PROTOCOL
  workfileeSZ5g4.if

GOAL
  %% see the HLPSL specification..
```

Risultato del Back-End TR4SP:

```
SUMMARY
   INCONCLUSIVE

DETAILS
   TIME_OUT

PROTOCOL
   /home/avispa/web-interface-
computation/./tempdir/workfileeSZ5g4.if

GOAL
   SECRECY

BACKEND
   TA4SP

COMMENTS
   Computation broken
```

4 Conclusioni

Lo sviluppo di protocolli di sicurezza veramente affidabili è molto complesso. A dispetto della loro appartente semplicità, infatti, i protocolli sono notoriamente soggetti ad errori. È complicato, anche mediante attente ispezioni del protocollo, determinare tutti i possibili scenari d'esecuzione. La valutazione della correttezza di un protocollo risulta, pertanto, molto difficile. È di estrema importanza, di conseguenza, disporre di strumenti che supportino l'analisi formale dei protocolli finalizzata alla ricerca di vulnerabilità o alla verifica della correttezza degli stessi. Negli ultimi anni sono stati proposti diversi strumenti automatici in grado di analizzare formalmente i protocolli. AVISPA ne è un esempio.

Lo scopo di questo lavoro ha consentito di effettuare l'analisi formale, tramite il tools AVISPA, del protocollo di Single Sign-On denominato OpenID Connect. Lo studio dell'OpenID Connect è iniziato dall'analisi della documentazione online. In un secondo momento il protocollo è stato formalizzato in notazione Alice-Bob e, a partire da tale notazione, è stata prodotta la specifica nel linguaggio HLPSL. L'analisi di quest'ultima, effettuata mediante i back-end di AVISPA, ha confermato, infine, come tale protocollo nella sua formulazione minimale, ovvero di default, sia soggetto a delle piccole vulnerabilità relative all'intercettazione di pacchetti contenenti informazioni di sessione, che possono essere rimosse implementando le funzionalità già presenti nelle specifiche dello stesso protocollo, come la crittografia e la firma digitale.

I Punti di forza del OpenID Connect si possono così riassumere:

- **Security and Privacy**: tali caratteristiche sono implementate attraverso l'uso di protocolli certificati quali: JSON Web Token (JWT), JSON Web Encryption (JWE), JSON Web Signature (JWS) e Transport Layer Security (TLS);
- **Ease of deployment, Flexibility and Wide support of devices**: la specifica si basa sull'uso di protocolli ampiamenti diffusi (p.e. OAuth 2.0, TLS, JSON e HTTP) e, pertanto, conformi ai goals indicati.
- **Interoperability and Enabling Claims Providers to be distinct from Identity Providers**: le informazioni sono scambiate direttamente ed in maniera sicura tra i soggetti interessati senza passaggi intermedi e sotto il controllo/consenso dell'utente finale.

Le specifiche del OpenId Connect consentono di usufruire di importanti contromisure per altri tipi di vulnerabilità, che questa analisi non ha potuto evidenziare per i limiti dello strumento utilizzato, come per esempio:

1. Il valore **nonce** del ID Token consente di associare una sessione Client con l'ID Token. Ciò permette di mitigare i **Replay Attacks**;
2. Il valore **state**, utilizzato per mantenere lo stato tra la richiesta e il callback, se viene vincolato crittograficamente con il valore di un cookie del browser, consente di mitigare gli attacchi cosidetti **Cross-Site Request Forgery (**CSRF, XSRF);
3. La **firma del Token** risolve la vulnerabilità conosciuta con il nome di **Token Manufacture/Modification** ,
4. L'uso della **crittografia** basata su chiavi simmetriche segrete (p.e. Client_secret) può risolvere il problema del **Server Response Disclosure**
5. Allo stesso modo l'utilizzo della **firma digitale** rimuove il problema della **Server Response Repudiation** e della **Request Repudiation**,
6. L'utilizzo del campo **AUD** (**Audience**) permette di eliminare la minaccia legata al riutilizzo dell'Access Token da parte di un Attaccante: **Access Token Redirect,**
7. La possibilità del **Token Reuse** è mitigata grazia all'utilizzo dei **timestamp** e di un valore di **life,**
8. La **Token Substitution** è mitigata grazie alle caratteristiche dell'**ID Token** implementate nel protocollo OpenId Connect.

E` molto importante, infine, continuare la ricerca nel campo dell'analisi formale dei protocolli di sicurezza. Lo studio di nuove tecniche, infatti, contribuirebbe ad allargare le possibilità degli strumenti automatici finora sviluppati. Alla luce dell'esperienza maturata a seguito di questo lavoro è emersa l'esigenza di una maggiore espressività del linguaggio HLPSL che permetta, in particolare, la modellazione di canali sicuri. Attualmente la loro rappresentazione si basa su approssimazioni non completamente soddisfacenti che potrebbero portare all'individuazione di falsi attacchi ai protocolli. Più in generale, dotare AVISPA di ulteriori back-end consentirebbe la specifica e la verifica di una classe più ampia di protocolli e loro proprietà rispetto a quelli attualmente considerati. Inoltre renderebbe possibile contemplare altri modelli di attacco. In questo modo l'elaborazione di nuovi protocolli potrebbe essere ulteriormente accelerata garantendo, al tempo stesso, l'affidabilità del prodotto finale.

Bibliografia

OpenID specifications: http://openid.net/specs

C. Caleiro, L. Viganò, D. Basin, On the Semantics of Alice&Bob Specifications of Security Protocols, Theoretical Computer Science. To appear.

A. Armando, R. Carbone, L. Compagna, J. Cuèllar, and M. Llanos Tobarra. Formal Analysis of SAML 2.0 Web Browser Single Sign-On: Breaking the SAML-based Single Sign-On for Google Apps. In Vitaly Shmatikov, editor, FMSE, pages 1–10. ACM, 2008.

A. Armando and L. Compagna. SAT-based Model-Checking for Security Protocols Analysis. Int. J. Inf. Sec., 7(1):3–32, 2008.

S. Modersheim D. A. Basin and L. Viganò. OFMC: A Symbolic Model Checker for Security Protocols. Int. J. Inf. Sec., 4(3):181–208, 2005.

The AVISPA Team. Deliverable D2.1: The High Level Protocol Specification Language, 2003. http://www.avispa-project.org/publications.html.

The AVISPA Team. AVISPA v1.1 User Manual, 2006. http://www. avispa-project.org/publications.html.

The AVISPA Team. HLPSL Tutorial: A Beginner's Guide to Modelling and Analysing Internet Security Protocols, 2006. http://www.avispa-project. org/publications.html.

M. Turuani. The CL-AtSe Protocol Analyser. In RTA, pages 277–286, 2006.

L. Vigano. Automated Security Protocol Analysis with the AVISPA Tool. Electr. Notes Theor. Comput. Sci., 155:61–86, 2006.

O. Heen Y. Glouche, T. Genet and O. Courtay. A Security Protocol Animator Tool for AVISPA. In ARTIST-2 workshop on security of embedded systems, Pisa (Italy), 2006.A. Armando, R. Carbone, L. Campagna, J. Cuellar, L. Tobarra Formal Analysis of SAML 2.0 Web Browser Single Sign-On: Breaking the SAML-based Single Sign-On for Google Apps. ACM Press (2008).