Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Miha Eržen

Odprta avtentikacija

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: doc. dr. Rok Rupnik

Ljubljana, 2012

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00252/2012
Datum: 05.04.2012

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MIHA ERŽEN**

Naslov: **ODPRTA AVTENTIKACIJA**
OPEN AUTHENTICATION

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Proučite in analizirajte model odprte avtentikacije ter opredelite in podrobno predstavite vse metode tega modela. Poseben poudarek dajte metodam, ki so potrebne za kriptiranje, digitalno podpisovanje in prenos podatkov med aplikacijo in strežnikom.

Mentor:
doc. dr. Rok Rupnik

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:
prof. dr. Nikolaj Zimic

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Miha Eržen, z vpisno številko **63060225**, sem avtor diplomskega dela z naslovom:

Odprta avtentikacija

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Roka Rupnika,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 11. oktobra 2012 Podpis avtorja:

Zahvaljujem se vsem, ki so mi bili v času študija v pomoč.

Kazalo

Seznam uporabljenih kratic in simbolov

Seznam uporabljenih kratic in simbolov, ki morajo biti enotni v celotnem delu, ne glede na označevanje v uporabljenih virih:
OAuth (ang.) Open Authentication
URI (ang.) Uniform resource identifier
API (ang.) Application programming interface
TLS (ang.) Transport Layer Security
SSL (ang.) Secure Sockets Layer

Povzetek

Diplomsko delo *Odprta avtentikacija* skuša bralcu podrobno predstaviti odprt protokol za avtorizacijo, s katerim bi lahko v prihodnosti odpravili oz. omilili potrebo po nenehnem vpisovanju gesel v spletne sisteme in druge aplikacije. S pomočjo odprte avtentikacije lahko uporabniki delijo svoje privatne resurse (datoteke, osebne informacije ipd.), shranjene na nekem strežniku z aplikacijo, ne da bi aplikaciji zaupali svoje uporabniško ime in geslo.

V delu je na začetku predstavljena kratka zgodovina odprte avtentikacije ter kratek opis razlike med tradicionalnim modelom avtentikacije (uporabniško ime in geslo) in odprto avtentikacijo. Predstavljena je tudi vloga lastnika resursov, ki je bila dodana tradicionalnemu modelu avtentikacije za potrebe avtorizacije. Na kratko so opisani osnovni principi delovanja odprte avtentikacije.

V nadaljevanju dela so podrobno predstavljene metode odprte avtentikacije. Opisan je tok dogodkov pri izmenjavi informacij med uporabnikom, aplikacijo in strežnikom ter vse metode, ki so potrebne za primerno kriptiranje, podpisovanje in prenos teh informacij med končnimi točkami. Na kratko so predstavljene tudi slabe lastnosti odprte avtentikacije. Kako so te težave rešene v verziji 2, je opisano v drugem delu diplomskega dela.

Ključne besede: odprta avtentikacija, avtorizacija, model avtentikacije

Abstract

The thesis *Open Authentication* aims to present in detail the open authorization protocol that could be used in the future to resolve or mitigate the need for constantly entering passwords into online systems and other applications. With open authentication users can share their private resources (files, personal information, etc.), which are stored on some server with the application, without having to provide the application with their user name and password.

The thesis begins with a short presentation of the history of open authentication, as well as a short description of the difference between the traditional model of authentication (user name and password) and open authentication. Also presented is the role of the resource owner, which has been added to the traditional model of authentication for the purposes of authorization. The basic principles of how open authentication functions are also briefly outlined.

The thesis continues with a detailed presentation of open authentication methods. It describes the chain of events in exchanging information between the user, the application, and the server, as well as all methods necessary for proper encryption, signing, and transferring this information between end points. The thesis also presents in brief the downsides to open authentication. The way these problems are resolved in version 2 is described in the second part of the thesis.

Keywords: open authentication, authorization, authentication model

Poglavje 1

Uvod

Zadnja leta je internet preplavilo ogromno spletnih aplikacij, portalov in družbenih omrežij, ki na vsakem koraku zahtevajo našo prijavo v njihove sisteme. S prvo prijavo moramo navadno vnesti še celo kopico naših bolj ali manj osebnih podatkov, preko katerih nas potem te aplikacije prepoznajo. Če si predstavljamo preprost scenarij uporabe teh spletnih aplikacij, lahko hitro ugotovimo, da gre za zelo moteč problem.

Posebej se stvari zapletejo, ko želimo deliti informacije, shranjene na nekem strežniku, z neko drugo spletno aplikacijo. Če želimo to doseči, moramo tej aplikaciji omogočiti dostop do naših osebnih informacij, resursov. V tem primeru bi morali s to aplikacijo deliti svoje uporabniško ime in geslo za dostop do strežnika.

Odprta avtentikacija odpravlja potrebo po delitvi svojih uporabniških imen in gesel z aplikacijami in storitvami na internetu.

Poglavje 2

Odprta avtentikacija

Začetki odprte avtentikacije (v nadaljevanju OAuth) segajo v leto 2006, ko so razvijalci podjetja Twitter razvijali svojo implementacijo takrat popularnega standarda za avtoriziranje imenovanega OpenID. Hoteli so združiti njihov programski vmesnik (v nadaljevanju API) s storitvijo OpenID za potrebe avtorizacije. Prišli so do zaključka, da na trgu ne obstaja nobena odprta storitev, ki bi ustrezala zahtevam in potrebam njihovega projekta.

Slika 2.1: OAuth logotip.

Aprila leta 2007 je bila ustanovljena manjša skupina implementatorjev, ki so spisali predlog za odprti protokol. Izkazalo se je, da težava ni značilna le za OpenID. Julija 2007 je ekipa sestavila osnutek specifikacije in Google skupina je postala odprta za vse, ki so bili zainteresirani za projekt oziroma so bili pripravljeni prispevati. Oktobra 2007 je bil izdan končni osnutek OAuth 1.0. Aprila 2010 je bil OAuth objavljen kot RFC 5849, od avgusta 2010 pa Twitter zahteva, da vse aplikacije drugih izdelovalcev uporabljajo standard OAuth.

Poglavje 3

Predstavitev

V tradicionalnem modelu avtentikacije (odjemalec-strežnik), odjemalec uporabi svoje poverilnice[1] za dostop do resursov na strežniku. Z naraščanjem spletnih storitev in računalništva v oblaku se je pokazala potreba, da bi bili ti resursi dostopni za programsko opremo drugega izdelovalca.

OAuth tradicionalnemu modelu avtentikacije (odjemalec-strežnik) doda še eno vlogo: lastnika resursov. V modelu OAuth odjemalec (ki ni lastnik resursov, ampak deluje v njegovem imenu) zahteva dostop do resursov, ki se nahajajo na strežniku. S temi resursi upravlja njihov lastnik. OAuth strežniku omogoča, da preveri avtorizacijo lastnika resursov, kot tudi identiteto odjemalca, ki podaja zahtevo.

OAuth odjemalcu zagotavlja metodo za dostop do resursov na strežniku v imenu lastnika resursov. Zagotavlja tudi postopek za končne uporabnike, da dovolijo tretji osebi (ali programski opremi drugega izdelovalca) dostop do njihovih resursov, shranjenih na strežniku, brez vnosa uporabniškega imena in gesla.

Da bi odjemalec prejel dostop do resursov, mora najprej prejeti dovoljenje lastnika resursov. To dovoljenje je predstavljeno v obliki žetona (token) in pripadajoče skrivnosti (shared-secret). Namen žetona je, da uporabniku ni potrebno deliti svojega uporabniškega imena in gesla z odjemalcem. Za

[1] Uporabniško ime in geslo.

razliko od tradicionalnih poverilnic (uporabniško ime in geslo) lahko žetonom določimo dovoljenja za dostop, jim omejimo življenjsko dobo in jih prekličemo neodvisno od poverilnic.

3.1 Primer

Jana (lastnica resursov) je pred kratkim na 'fotografije.primer.si' (strežnik) naložila zasebne fotografije s potovanja po Indiji (privatni resursi). Rada bi uporabila spletno aplikacijo 'natisni.primer.com' (odjemalec) za tiskanje nekaterih fotografij s potovanja. Za prijavo na strežnik Jana uporabi svoje uporabniško ime in geslo.

Vendar pa Jana ne želi deliti svojega uporabniškega imena in gesla s spletno stranjo 'natisni.primer.com', ki rabi dostop do njenih fotografij na strežniku, da jih lahko natisne. Da bi svojim uporabnikom ponudili boljše storitve, je spletna stran 'natisni.primer.com' pridobila poverilnice za odjemalca od spletne strani 'fotografije.primer.si'.

```
Odjemalčev identifikator:
dpf43f3p2l4k3l03

Odjemalčeva deljena skrivnost:
kd94hf93k423kf44
```

Spletna stran 'natisni.primeri.com' je tudi nastavila svojo aplikacijo tako, da uporablja končne točke protokola, kot je opisano v dokumenticaiji aplikacije 'fotografije.primer.si'.

Zahteva za začasne poverilnice:

```
https://fotografije.primer.si/initiate
```

Url za avtorizacijo lastnika resursov:

```
https://fotografije.primer.si/authorize
```

Url za zahtevo žetona:

```
https://fotografije.primer.si/token
```

Preden pa lahko spletna stran 'natisni.primeri.com' od Jane zahteva dovoljenje za dostop do slik, mora najprej vzpostaviti sklop začasnih poverilnic s spletno stranjo 'fotografije.primer.si' za identifikacijo zahtevka. Da to doseže, odjemalec pošlje strežniku naslednji HTTPS zahtevek:

```
POST /initiate HTTP/1.1
Host: fotografije.primer.si
Authorization: OAuth realm="Slike",
    oauth_consumer_key="dpf43f3p2l4k3l03",
    oauth_signature_method="HMAC-SHA1",
    oauth_timestamp="137131200",
    oauth_nonce="wIjqoS",
    oauth_callback="http%3A%2F%2Fnatisni.primer.com%2Fready",
    oauth_signature="74KNZJeDHnMBp0EMJ9ZHt%2FXKycU%3D"
```

Strežnik potrdi zahtevo in odgovori s sklopom začasnih poverilnic v telesu HTTP-odgovora.

```
HTTP/1.1 200 OK
Content-Type: application/x-www-form-urlencoded

oauth_token=hh5s93j4hdidpola&oauth_token_secret=hdhd0244k9j7ao03&
oauth_callback_confirmed=true
```

Odjemalec nato preusmeri Janin brskalnik na končno točko lastnika resursov, kjer pridobi Janino dovoljenje za dostop do njenih zasebnih fotografij.

```
https://fotografije.primeri.si/authorize?oauth_token=hh5s93j4hdidpola
```

Strežnik od Jane nato zahteva, da vpiše svoje uporabniško ime in geslo. Če je prijava uspešna, jo prosi še za dovoljenje za dostop do njenih zasebnih fotografij v imenu odjemalca (natisni.primeri.com). Ko Jana dovoli dostop

do svojih fotografij, je njen spletni brskalnik preusmerjen na povezavo, ki pričakuje odgovor od strežnika:

```
http://natisni.primer.com/ready?
    oauth_token=hh5s93j4hdidpola&oauth_verifier=hfdp7dh39dks9884
```

Povratni klic pove odjemalcu, da je Jana zaključila proces avtorizacije. Odjemalec nato zahteva sklop žetonov z uporabo začasnih poverilnic:

```
POST /token HTTP/1.1
Host: fotografije.primer.si
Authorization: OAuth realm="Fotografije",
oauth_consumer_key="dpf43f3p2l4k3l03",
oauth_token="hh5s93j4hdidpola",
oauth_signature_method="HMAC-SHA1",
oauth_timestamp="137131201",
oauth_nonce="walatlh",
oauth_verifier="hfdp7dh39dks9884",
oauth_signature="gKgrFCywp7rO0OXSjdot%2FIHF7IU%3D"
```

Strežnik nato potrdi zahtevo in odgovori s sklopom žetonov:

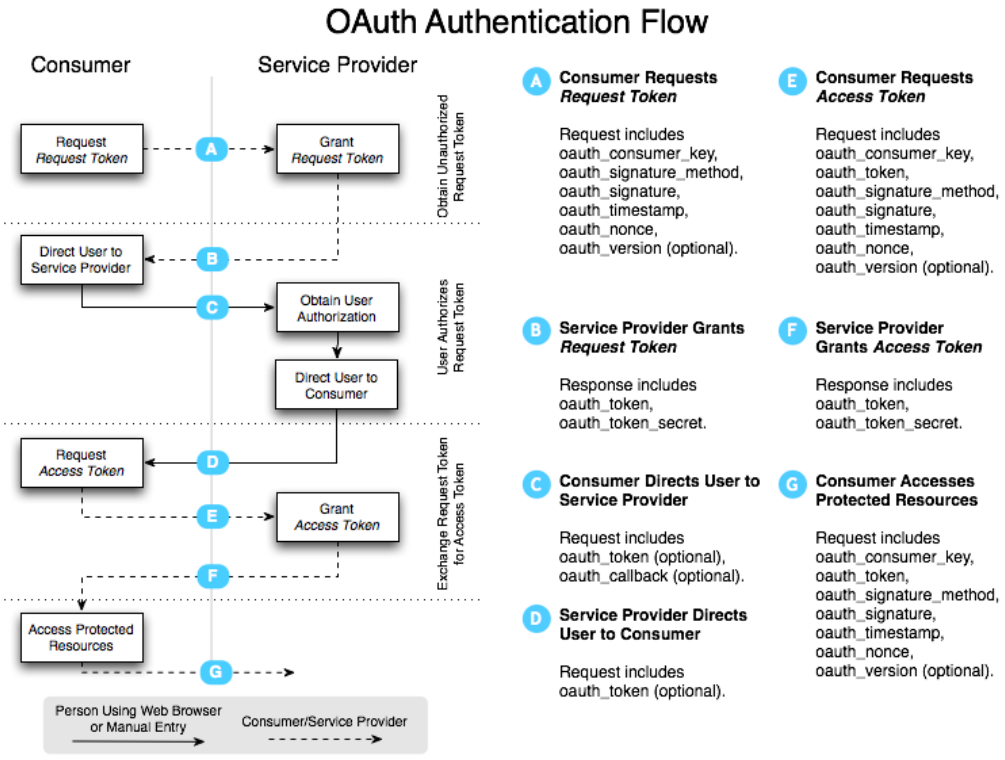
```
HTTP/1.1 200 OK
Content-Type: application/x-www-form-urlencoded

oauth_token=nnch734d00sl2jdk&oauth_token_secret=pfkkdhi9sl3r4s00
```

S tem sklopom žetonov lahko sedaj odjemalec poda zahtevo za zasebno fotografijo:

```
GET /fotografije?datoteka=pocitnice.jpg&velikost=original HTTP/1.1
Host: fotografije.primer.si
Authorization: OAuth realm="Fotografije",
oauth_consumer_key="dpf43f3p2l4k3l03",
oauth_token="nnch734d00sl2jdk",
```

```
oauth_signature_method="HMAC-SHA1",
oauth_timestamp="137131202",
oauth_nonce="chapoH",
oauth_signature="MdpQcU8iPSUjWoN%2FUDMsK2sui9I%3D"
```

Strežnik 'fotografije.primer.si' potrdi zahtevo in odgovori z zahtevano sliko. Spletna stran 'natisni.primer.com' lahko še naprej uporablja Janine zasebne slike z istim sklopom žetonov, vse dokler Jana omenjenim žetonom ne prekliče dostopa.



Slika 3.1: OAuth tok dogodkov.

Poglavje 4

Avtorizacija na podlagi preusmeritev

OAuth uporablja žetone, ki predstavljajo, da je odjemalec prejel pravice od lastnika resursov. Žetone tipično izda strežnik, ko to odjemalec od njega zahteva in ko je strežnik že potrdil preverjeno prisotnost lastnika resursov (z uporabo uporabniškega imena in gesla).

Obstaja veliko načinov, s katerimi strežnik olajša rezervacijo žetonov. Ena izmed teh je avtorizacija na podlagi preusmeritev. Ta metoda vsebuje tri korake:

1. Odjemalec od strežnika pridobi sklop začasnih poverilnic, ki se uporabijo za identifikacijo zahtevka v celotnem postopku avtorizacije.

2. Lastnik resursov pooblasti strežnik, da odjemalcu dovoli pošiljanje zahtevkov.

3. Odjemalec nato z uporabo začasnih poverilnic od strežnika prejme žeton, s katerim lahko nato dostopa do datotek, ki so v lasti lastnika resursov.

Strežnik mora začasne poverilnice preklicati, ko so bile uporabljene za pridobitev žetona. Priporočljivo je tudi, da imajo začasne poverilnice omejeno

življenjsko dobo. Strežnik pa naj bi lastniku resursov omogočal, da prekliče žetone, ki so bili dodeljeni odjemalcu.

Da lahko odjemalec izvede te tri korake, mora strežnik oglaševati URI-je naslednjih končnih točk:

- Zahtevek za začasne poverilnice
- Avtorizacija lastnika resursov
- Zahtevek za žeton

4.1 Začasne poverilnice

Odjemalec od strežnika pridobi sklop začasnih poverilnic tako, da pošlje avtenticiran HTTP POST-zahtevek na končno točko za začasne poverilnice. Odjemalec sestavi URI-zahtevek tako, da doda zahtevan parameter oatuh_callback[1]. Strežnik lahko specificira tudi druge parametre.

Ker se odgovor na zahtevek prenaša kot golo besedilo, mora strežnik zahtevati uporabo kriptirane povezave, npr. TLS[2] ali SSL.

Primer: odjemalec odda naslednji zahtevek:

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

Strežnik mora zahtevek preveriti in če je veljaven, nanj odgovoriti s sklopom začasnih poverilnic (v obliki identifikatorja in deljene skrivnosti). Začasne poverilnice so vključene v telesu HTTP-odgovora.

[1]Parameter oatuh_callback je URI, kamor strežnik preusmeri odjemalca po končanem drugem koraku avtorizacije (avtorizacija lastnika resursov).

[2]Kriptografski protokol, ki poskrbi za varen prenos informacij po internetu.

Odgovor vsebuje naslednje zahtevane parametre:

- oauth_token - Začasni identifikator poverilnic.
- oauth_token_secret - Začasna deljena skrivnost poverilnic.
- oauth_callback_confirmed - Mora biti prisoten in nastavljen na true. Uporablja se ga za razločevanje od prejšnjih verzij protokola.

Kljub temu da imena parametrov vsebujejo besedo token (žeton), te poverilnice niso žetoni.

4.2 Avtorizacija lastnika resursov

Preden odjemalec od strežnika zahteva sklop žetonov, mora uporabnika preusmeriti na strežnik, kjer uporabnik zahtevek avtorizira. Odjemalec nato sestavi URI tako, da končni točki za avtorizacijo lastnika resursov doda zahtevan parameter oauth_token[3]. Tudi tukaj lahko strežnik zahteva dodatne parametre.

Odjemalec lastnika resursov nato preusmeri na sestavljen URI z uporabo HTTP-preusmeritve. Zahtevek mora uporabiti HTTP GET-metodo.

Po prejemu avtorizacijske odločitve lastnika resursov ga strežnik preusmeri na URI, ki je bil posredovan preko parametra (oauth_callback) ali na kakšen drugi način.

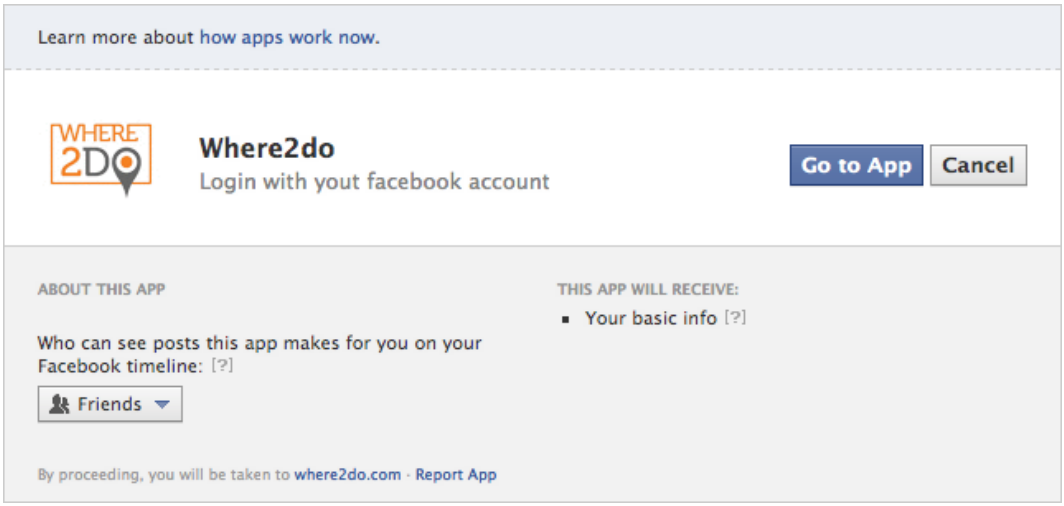
Za zagotovitev, da je lastnik resursov, ki je dovolil dostop, isti kot lastnik resursov, ki se vrača nazaj k odjemalcu za zaključek procesa, mora strežnik generirati kodo za preverjanje[4]. Strežnik sestavi URI za zahtevek tako, da doda zahtevane parameter na URI za povratni klic:

- oauth_token - Začasni identifikator poverilnic prejet s strani odjemalca.
- oauth_verifier - Koda za preverjanje.

[3]Identifikator začasnih poverilnic, pridobljen v prejšnjem koraku.

[4]Neuganljivo zaporedje znakov, ki se ga prenese k odjemalcu preko lastnika resursov in je zahtevano za končanje procesa avtorizacije.

Če odjemalec ni priskrbel URI-ja za povratni klic, bi moral strežnik nato prikazati vrednost kode za preverjanje in lastniku resursov naročiti, da to kodo sporoči odjemalcu za zaključek avtorizacije.



Slika 4.1: Okno facebook za avtorizacijo aplikacije.

4.3 Žetoni

Odjemalec od strežnika pridobi set žetonov tako, da pošlje avtenticirano HTTP POST-zahtevo na končno točko za zahtevke za žetone. Odjemalec sestavi URI za zahtevo s tem, da zahtevku doda zahtevan parameter (oauth_verifier).

Ob oddajanju zahtevka se odjemalec avtenticira z odjemalčevimi poverilnicami kot tudi z začasnimi poverilnicami. Začasne poverilnice se uporabljajo namesto žetonov v zahtevku in se prenašajo z uporabo 'oauth_token' parametra.

Ker se odgovor na zahtevek prenaša kot golo besedilo, mora strežnik zahtevati uporabo kriptirane povezave npr. TLS ali SSL. Primer:

```
POST /request_token HTTP/1.1
Host: server.example.com
```

```
Authorization: OAuth realm="Example",
oauth_consumer_key="jd83jd92dhsh93js",
oauth_token="hdk48Djdsa",
oauth_signature_method="PLAINTEXT",
oauth_verifier="473f82d3",
oauth_signature="ja893SD9%26xyz4992k83j47x0b"
```

Strežnik mora zahtevek preveriti, zagotoviti, da je lastnik resursov odjemalcu dovolil rezervacijo žetonov in zagotoviti, da so začasne poverilnice še vedno veljavne oz. še niso bile uporabljene. Strežnik mora preveriti tudi kodo za preverjanje, ki jo je prejel od odjemalca. Če je zahtevek veljaven in avtenticiran, potem so žetoni vključeni v HTTP-odgovoru. Odgovor vsebuje naslednje zahtevane parametre:

- oauth_token - identifikator žetona
- oatuh_token_secret - deljena skrivnost žetona

Primer:

```
HTTP/1.1 200 OK
Content-Type: application/x-www-form-urlencoded
oauth_token=j49ddk933skd9dks&oauth_token_secret=ll399dj47dskfjdk
```

Ko odjemalec prejme in shrani žetone, lahko nadaljuje z dostopanjem do zaščitenih resursov v imenu lastnika resursov. To stori tako, da pošlje avtenticirane zahtevke z uporabo odjemalčevih poverilnic skupaj z žetoni, ki jih je prejel.

Poglavje 5

Avtenticirani zahtevki

Odjemalci z uporabo avtenticiranih zahtevkov pridobijo dostop do zaščitenih resursov tako, da uporabijo njihove poverilnice (tipično uporabniško ime in geslo). To strežniku omogoča, da preveri in potrdi njihovo avtentičnost. Uporaba teh metod zahteva, da se odjemalca predstavlja v vlogi lastnika resursov.

OAuth vsebuje metodo, ki vključuje dva seta poverilnic z vsakim zahtevkom. En set za identificiranje odjemalca in drugi za identificiranje lastnika resursov. Preden lahko odjemalec odda avtenticiran zahtevek v imenu lastnika resursov, mora najprej pridobiti avtenticiran žeton od lastnika resursov.

5.1 Oddajanje zahtevkov

Avtenticiran zahtevek vsebuje različne parametre protokola. Imena teh parametrov se začnejo s predpono 'oauth_' (imena in vrednosti so občutljivi na velike in male črke). Odjemalci oddajajo zahtevke v naslednjih korakih:

1. Odjemalec nastavi vrednosti vsakemu od naslednjih zahtevanih parametrov protokola:

    - oauth_consumer_key - Identifikator poverilnic odjemalca (ekvivalentno uporabniškemu imenu).

- oauth_token - Vrednost žetona, ki se uporablja za povezovanje zahtevka z lastnikom resursa.
- oauth_signature_method - Metoda podpisa, ki jo uporablja odjemalec za podpisovanje zahtevka.
- oauth_timestamp - Časovni žig.
- oauth_nonce - Naključni niz besed.
- oauth_version - Opcijski parameter. Pove verzijo protokola, ki se uporablja.

2. Parametri protokola se dodajo k zahtevku z uporabo ene izmed prenosnih metod. Vsak parameter se lahko v zahtevku pojavi le enkrat.
3. Odjemalec izračuna in dodeli vrednost 'oauth_signature' parametru in doda parameter v zahtevek z uporabo prenosnih metod.
4. Odjemalec pošlje zahtevek na strežnik.

Primer za oddajo HTTP-zahtevka:

```
POST /request?b5=%3D%253D&a3=a&c%40=&a2=r%20b HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded

c2&a3=2+q
```

Odjemalec dodeli vrednost naslednjim parametrom protokola z uporabo svojih poverilnic, z žetonom, z uporabo časovnega žiga, unikatnega niza znakov in označuje, da bo uporabil HMAC-SHA1-metodo podpisovanja:

```
oauth_consumer_key:     9djdj82h48djs9d2
oauth_token:            kkk9d7dh3k39sjv7
oauth_signature_method: HMAC-SHA1
oauth_timestamp:        137131201
oauth_nonce:            7d8f3e4a
```

Odjemalec doda parametre protokola zahtevku:

```
Authorization: OAuth realm="Example",
oauth_consumer_key="9djdj82h48djs9d2",
oauth_token="kkk9d7dh3k39sjv7",
oauth_signature_method="HMAC-SHA1",
oauth_timestamp="137131201",
oauth_nonce="7d8f3e4a"
```

Odjemalec nato izračuna vrednost 'oauth_signature' parametra (z uporabo skrivnosti in žetona), ga doda zahtevku in pošlje zahtevek na strežnik:

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

c2&a3=2+q
```

5.2 Preverjanje zahtevkov

Strežnik mora prejete zahtevke potrditi z naslednjimi koraki:

1. Ponovno mora neodvisno izračunati podpis zahtevka in ga primerjati s prejetim podpisom s strani odjemalca z uporabo oauth_signature-parametra.

2. Zagotoviti mora, da je kombinacija naključnega niza znakov, časovnega žiga in žetona, prejeta od odjemalca, unikatna in še ni bila uporabljena v prejšnjih zahtevkih. Strežnik lahko zastarele časovne žige zavrne.

3. Če je žeton prisoten, mora strežnik potrditi obseg in status odjemalčeve avtorizacije, kot je predstavljeno v žetonu.

V primeru, da zahtevek ni bil potrjen, naj bi strežnik nanj odgovoril z ustrezno HTTP-status kodo. Strežnik lahko v odgovor vključi tudi druge podrobnosti o tem, zakaj je bil zahtevek zavrnjen.

Strežnik naj bi odgovoril s kodo 400 (slaba zahteva), kadar:

- prejme zahtevek z nepodprtimi parametri,
- prejme zahtevek z nepodprto metodo za podpisovanje,
- manjkajo parametri ali
- se parametri podvajajo.

Strežnik naj bi odgovoril s kodo 401 (nepooblaščen), kadar:

- prejme zahtevek z neveljavnimi odjemalčevimi poverilnicami,
- prejme zahtevek z neveljavnim ali poteklim žetonom,
- prejme zahtevek z neveljavnim podpisom,
- prejme zahtevek z neveljavnim ali že uporabljenim nizom naključnih znakov.

5.3 Naključni niz in časovni žig

Vrednost časovnega žiga mora biti pozitivno celo število. Če ni drugače specificirano, potem mora biti časovni žig predstavljen kot število sekund, ki so pretekle od 1. januarja 1970 00:00:00 GMT.

Naključni niz je unikaten niz naključnih znakov, ki jih generira odjemalec. Uporablja se za preverjanje, da zahtevek prej še ni bil poslan in onemogoča napade, kadar so zahtevki poslani preko nekriptirane povezave. Niz mora biti unikaten v vseh zahtevkih z isto kombinacijo časovnega žiga, odjemalčevih poverilnic in žetonom.

5.4 Podpis

Zahtevki, avtenticirani z OAuth, imajo lahko dva seta poverilnic: poslane preko oauth_consumer_key-parametra in tiste v oauth_token-parametru. Da lahko strežnik potrdi pristnost zahtevka in prepreči neavtoriziran dostop, mora odjemalec dokazati, da je zakoniti lastnik poverilnic. To se doseže z uporabo deljene skrivnosti (RSA-ključ) v setu poverilnic.

OAuth zagotavlja tri metode za zagotavljanje pristnosti odjemalca:

1. HMAC-SHA1
2. RSA-SHA1
3. PLAINTEXT

Te metode imenujemo metode za podpisovanje, čeprav 'PLAINTEXT' ne vsebuje podpisa. Odjemalec pove, katera metoda je uporabljena z 'oauth_signature_method' parametrom.

5.5 Prenos parametrov

Pri izvajanju OAuth-avtenticiranega zahtevka, so parametri protokola kot tudi kateri koli drugi parametri s predpono 'oauth_' vključeni v zahtevek z uporabo ene in samo ene od naštetih lokacij (prva ima večjo prednost kot druga itn.):

1. HTTP-polje za avtorizacijo ('Authorization') v glavi zahtevka

2. HTTP-telo entitete

3. HTTP URI-poizvedba

5.5.1 Polje za avtorizacijo

Parametri protokola se lahko prenašajo preko polja za avtorizacijo v glavi zahtevka. Primer:

```
Authorization: OAuth realm="Example",
oauth_consumer_key="0685bd9184jfhq22",
oauth_token="ad180jjd733klru7",
oauth_signature_method="HMAC-SHA1",
oauth_signature="wOJIO9A2W5mFwDgiDvZbTSMK%2FPY%3D",
oauth_timestamp="137131200",
oauth_nonce="4572616e48616d6d65724c61686176",
oauth_version="1.0"
```

Protokoli parametra so v glavo vključeni na naslednji način:

1. Imena in vrednosti parametrov.

2. Vsakemu imenu parametra takoj sledi znak =, nato znak ", vrednost parametra in nato še en znak ".

3. Parametri so med seboj ločeni z znakom ',' in opcijsko s presledkom.

5.5.2 Telo entitete

Parametri protokola se lahko prenašajo v telesu entitete, vendar samo, če so izpolnjeni naslednji pogoji:

1. Telo entitete je enodelno.

2. Telo entitete je kodirano po principu 'application/x-www-form-urlencoded'.

3. V glavi HTTP-zahtevka je vključeno polje 'Content-Type' in je nastavljeno na 'application/x-www-form-urlencoded'.

Primer:

```
oauth_consumer_key=0685bd9184jfhq22
&oauth_token=ad180jjd733klru7
&oauth_signature_method=HMAC-SHA1
&oauth_signature=wOJIO9A2W5mFwDgiDvZbTSMK%2FPY%3D
&oauth_timestamp=137131200
&oauth_nonce=4572616e48616d6d65724c61686176
&oauth_version=1.0
```

Telo entite lahko vključuje tudi druge parametre, vendar morajo biti v tem primeru parametri protokola vključeni za parametri zahtevka in pravilno ločeni z znakom '&'.

5.5.3 URI-poizvedba

Parametri protokola se lahko prenašajo v URI poizvedbi. Primer:

```
GET /example/path?oauth_consumer_key=0685bd9184jfhq22
&oauth_token=ad180jjd733klru7
&oauth_signature_method=HMAC-SHA1
&oauth_signature=wOJIO9A2W5mFwDgiDvZbTSMK%2FPY%3D
&oauth_timestamp=137131200
&oauth_nonce=4572616e48616d6d65724c61686176
&oauth_version=1.0 HTTP/1.1
```

URI za zahtevek lahko vključuje tudi druge parametre, vendar morajo biti v tem primeru parametri protokola vključeni za parametri zahtevka in pravilno ločeni z znakom '&'.

Poglavje 6

OAuth 2.0

6.1 Zakaj nova verzija

OAuth 1.0 je baziral na dveh takrat obstoječih protokolih:

- Flickr API-Auth
- Google AuthSub

Z več kot 3-letnimi izkušnjami dela na protokolu OAuth so se razvijalci naučili dovolj, da so prišli do ugotovitev, kako protokol izboljšati.

Tri glavna področja, na katerih se je protokol OAuth 1.0 izkazal za omejenega oz. je včasih povzročal zmedo:

1. avtentikacija in podpisi,
2. uporabniška izkušnja in možnost izdaje alternativnih žetonov,
3. skalabilnost in zmogljivost.

6.1.1 Avtentikacija in podpisi

Kompleksnost kriptografskih zahtev v specifikaciji za OAuth 1.0 je povzročila, da je veliko implementacij propadlo oz. niso bile izvedene v celoti.

Slika 6.1: OAuth 2.0 logotip.

Vzrok za to je lahko ta, da je bila prvotna specifikacija slabo spisana. Kasneje so spisali novo (RFC 5849), ki kompleksnosti še vedno ni odpravila. Priročnost in enostavnost uporabe, kot jo ima tradicionalni model avtentikacije (uporabniško ime in geslo), sta OAuth 1.0 zagotovo manjkala.

Razvijalci lahko hitro spišejo programe z uporabo uporabniškega imena in gesla. S selitvijo na OAuth pa so ti razvijalci prisiljeni v iskanje, nameščanje in konfiguracijo knjižnic, da dosežejo to, kar je bilo prej izvedljivo z eno vrstico cURL[1]-kode.

6.1.2 Uporabniška izkušnja

OAuth je sestavljen iz dveh glavnih delov:

- pridobitev žetona s privolitvijo lastnika resursov in
- uporaba žetona za dostop do zasebnih resursov.

Metode za pridobitev žetona za dostop imenujemo tokovi. OAuth 1.0 je bil sprva sestavljen iz treh takih tokov:

- spletne aplikacije,
- računalniške aplikacije (odjemalci) in
- mobilne aplikacije.

[1]Orodje ukazne vrstice za prenašanje informacij z URL-sintakso.

V razvoju specifikacije OAuth 1.0 so se ti trije tokovi združili v enega, ki naj bi zajel vse tri omenjene odjemalce. V praksi se je izkazalo, da tok sicer deluje za spletne aplikacije, vendar pri ostalih dveh ponuja slabšo uporabniško izkušnjo.

Ko je vse več strani začelo uporabljati OAuth (predvsem Twitter), so razvijalci spoznali, da se je ta poenoten tok izkazal za precej omejenega in ponavadi ponuja slabo uporabniško izkušnjo. Po drugi strani pa je Facebook Connect[2] ponujal bogat nabor tokov tako za spletne aplikacije kot tudi za mobilne naprave in igralne konzole.

6.1.3 Skalabilnost in zmogljivost

Ko so veliki ponudniki začeli uporabljati OAuth, so razvijalci prišli do ugotovitve, da protokol ni dovolj skalabilen. Zahteva upravljanje stanj po različnih korakih, začasne poverilnice so večkrat kot ne zavržene neuporabljene in tipično zahteva izdajo dolgo trajajočih poverilnic, ki so manj varne in jih je težje upravljati (in sinhronizira preko več centrov za hrambo podatkov).

Poleg tega OAuth 1.0 zahteva, da imajo končne točke za dostop do zasebnih resursov dostop do poverilnic uporabnika za potrditev zahteve. To polomi tipično arhitekturo velikih ponudnikov, ki imajo ponavadi en centralni strežnik za avtorizacijo poverilnic in ločen strežnik za API-klice. OAuth 1.0 zahteva rabo obeh sklopov poverilnic: poverilnice uporabnika in žetone, kar to ločitev (strežnikov) precej oteži.

6.2 Novosti

Novosti v OAuth 2.0 so:

- Tok uporabniškega vmesnika - za odjemalce, ki tečejo znotraj uporabniškega vmesnika (tipično je to spletni brskalnik).

[2]API, ki omogoča prijavo uporabnikov Facebooka v aplikacije tretje osebe.

- Tok spletnega strežnika - za odjemalce, ki so del strežniške aplikacije, dostopne preko HTTP-zahtev. To je poenostavljena verzija toka iz OAuth 1.0.
- Tok naprave - primeren za odjemalce, ki tečejo na omejenih napravah, končni uporabnik pa ima ločen dostop do brskalnika na drugem računalniku ali napravi.
- Tok uporabniškega imena in gesla - se uporablja v primerih, ko uporabnik zaupa odjemalcu. Hramba teh poverilnic še vedno ni zaželena. Ta tok je primeren samo, kadar obstaja veliko zaupanje med uporabnikom in odjemalcem.
- Tok odjemalčevih poverilnic - odjemalec uporabi svoje poverilnice za pridobitev žetonov. Ta tok podpira tako imenovani scenarij dveh nog (2-legged scenario).
- Tok uveljavljanja - odjemalec predstavi strežniku tako imenovano uveljavitev, na primer SAML[3], v zameno za žeton.

6.2.1 Nosilec žetonov

OAuth 2.0 zagotavlja opcijo, pri kateri ni potrebna uporaba kriptografije za avtenticiranje in temelji na obstoječi arhitekturi za avtenticiranje. Namesto pošiljanja zahtevkov podpisanih z HMAC in skrivnosti poverilnic je kar žeton sam uporabljen kot skrivnost poslana preko HTTPS. To omogoča uporabo cURL in ostalih enostavnih skriptnih orodij.

6.2.2 Poenostavljeni podpisi

Podpisovanje v OAuth 2.0 je bistveno poenostavljeno za odstranitev potrebe po dodatnem razčlenjevanju, kodiranju in sortiranju parametrov. Uporablja tudi samo eno skrivnost namesto dveh.

[3]Odprt XML-standard za izmenjavo podatkov o preverjanju prisotnosti med ponudnikom identitete in ponudnikom storitve.

6.2.3 Krajša življenjska doba žetonov

Namesto izdajanja žetonov z dolgo življenjsko dobo (eno leto ali celo neomejeno) lahko strežnik izda žeton s kratko življenjsko dobo in žeton za osveževanje z daljšo življenjsko dobo. To omogoča odjemalcu pridobitev novega žetona za dostop brez ponovnega vključevanja uporabnika v proces in hkrati ohranja žetone omejene.

Poglavje 7

Podobni sistemi in razlike

7.1 OpenID

OpenID je odprto, decentralizirano orodje, namenjeno digitalnim identitetam. Omogoča uporabo obstoječega računa za prijavo v različne spletne strani, ne da bi bilo potrebno ponovno ustvarjati novo geslo. Omogoča izbiro informacij, ki se povežejo z OpenID-računom, za deljenje z neko spletno stranjo (ime, e-pošta). Z OpenID je možna kontrola o tem, koliko in katere informacije se delijo z določeno spletno stranjo.

Slika 7.1: OpenID-logotip.

OpenID-standard določa, kako mora potekati komunikacija med izdajateljem identitete in prejemnikom OpenID-identitete na drugi strani. Dodatek k OpenID-standardu (OpenID Attribute Exchange) olajša prenos uporabnikovih atributov (ime in priimek, spol ...) do prejemnika identitete (vsak prejemnik lahko določi različen set atributov).

Ker je OpenID decentraliziran, se ne zanaša na centralno avtoriteto za avtentikacijo uporabnikove identitete. OpenID uporabljajo in zagotavljajo

podporo številne velike spletne strani:

- Google
- Yahoo!
- PayPal
- BBC
- AOL
- IBM
- Steam
- VeriSign

7.1.1 Razlike v primerjavi z OAuth

OpenID omogoča uporabniku uporabo enega uporabniškega imena in gesla za dostop do ostalih spletnih strani. Vsakič, ko se uporabnik želi prijaviti v spletno mesto, ki omogoča prijavo z OpenID, bo preusmerjen na OpenID-stran in nato nazaj.

OAuth omogoča uporabniku, da avtorizira določeno spletno mesto za dostop do podatkov na nekem strežniku. Ko hoče uporabik spletnemu mestu dovoliti pravice za dostop do podatkov, ga bo to spletno mesto preusmerilo na spletno mesto izadjatelja OAuth-poverilnic. Ko uporabnik odobri dostop do podatkov, je preusmerjen nazaj na prvotno stran.

7.2 Facebook connect

Facebook connect je skupek vmesnikov za programiranje, ki omogoča uporabnikom Facebooka prijavo v ostale spletne strani, aplikacije, mobilne naprave

in igralniške sisteme z uporabo njihove Facebook-identitete. Facebook connect je bil predstavljen javnosti julija 2008, dostopen pa je postal decembra 2008.

Facebook connect je rešitev za integracijo uporabnika na neko spletno mesto. Najbolj preprost način uporabe Facebook connecta je lažja registracija uporabnika v sistem.

Poleg lažje registracije uporabnika v sistem Facebook connect omogoča, da spletne strani razširijo ponudbo funkcij socialnega omrežja. To omogoča uporabniku, da lažje deli vsebino preko Facebooka.

Slika 7.2: Primer Facebook connect-okna.

7.2.1 Razlike v primerjavi z OAuth

Facebook connect je na voljo samo uporabnikom Facebooka in ga v taki obliki ni mogoče implementirati v lasten sistem. Facebook connect ni na voljo uporabnikom, ki dostopa do Facebooka nimajo (npr. Kitajska), čeprav je stran, ki zahteva avtorizacijo Facebook, dostopna.

Poglavje 8

Primer implementacije OAuth

OAuth je možno implementirati v vseh programskih jezikih, če se upoštevajo osnovne zakonitosti standarda. Zaradi razširjenosti programskega jezika PHP je primer implementacije protokola OAuth predstavljen v tem jeziku.

8.1 Opis primera

Za spletno aplikacijo želimo razviti sistem za prijavo, ki bo uporabljal obstoječ uporabniški račun uporabnika na popularnem socialnem omrežju. S tem želimo uporabniku prihraniti čas pri vnašanju njegovih podatkov. Zaradi razširjenosti in urejenosti njihove dokumentacije smo se odločili uporabiti Twitter.

Uporabnik bo na vstopni strani aplikacije predstavljen z možnostjo prijave preko socialnega omrežja. S klikom na gumb za prijavo bo uporabnik preusmerjen na Twitter, kjer bo obveščen o tem, katere podatke želimo od njega pridobiti.

Ko bo uporabnik odobril dostop do svojih podatkov, bomo s strežnika Twitter prenesli njegovo ime in njegovo profilno sliko ter mu izrekli dobrodošlico.

8.2 Tok dogodkov

Za uspešno implementacijo je potrebno od Twitterja pridobiti identifikator poverilnic (consumer_key) in skrivnost (consumer_secret) za odjemalca.

Te poverilnice pridobimo tako, da se logiramo na spletno stran http://dev.twitter.com in na njej ustvarimo novo aplikacijo z imenom, opisom, URL do spletne strani, na kateri bo aplikacija na voljo, do katerih uporabnikovih podatkov bo aplikacija dostopala in končne točke aplikacije (callback URL).

Po uspešni pridobitvi omenjenih poverilnic nadaljujemo implementacijo v naslednjih korakih:

1. Ustvarimo TwitterOAuth-objekt z uporabo odjemalčevih poverilnic.
2. Od Twitterja zahtevamo začasne poverilnice.
3. Ustvarimo URL za avtorizacijo s Twitterjem.
4. Preusmerimo uporabnika na ustvarjen URL.
5. Uporabnik potrdi dostop in se vrne na prvotno stran.
6. Ponovno ustvarimo TwitterOAuth-objekt z uporabnikovimi poverilnicami in začasnimi poverilnicami (client_credentials in temporary_credentials).
7. Pridobimo žeton od Twitterja.
8. Ponovno ustvarimo TwitterOAuth-objekt z uporabnikovimi poverilnicami in žetonom.
9. Od Twitterja zahtevamo podatke o uporabniku.

8.2.1 Uporabnik pride na stran

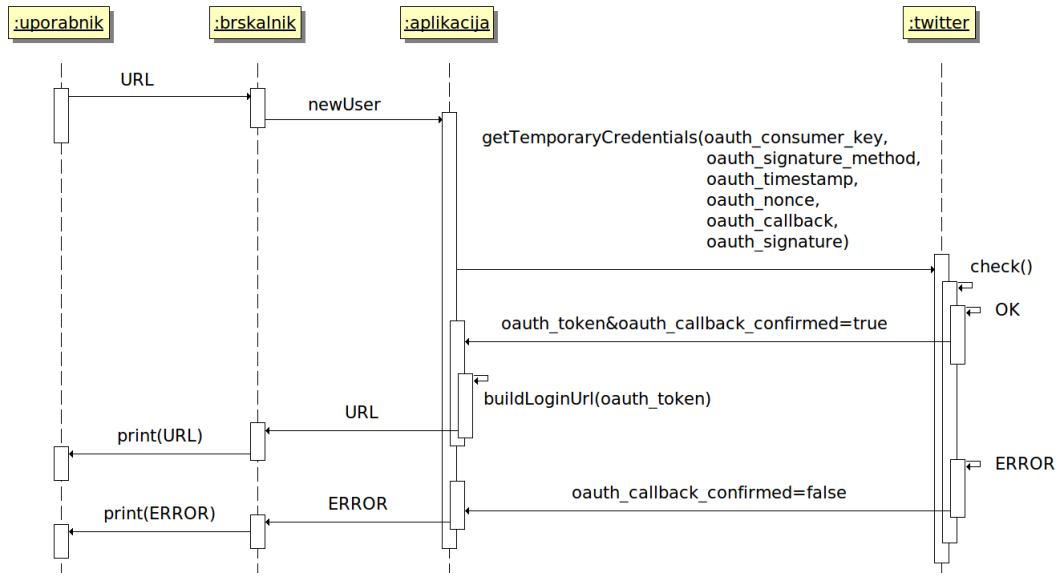
Ko uporabnik pride na stran, je potrebno kreirati URL za avtorizacijo s Twitterjem z uporabo poverilnic, ki smo jih predhodno pridobili od Twitterja.

Uporabnik v spletni brskalnik vnese URL-naslov aplikacije. Brskalnik aplikaciji sporoči, da je na strani nov uporabnik. Aplikacija od Twitterja

z uporabo pridobljenih poverilnic (consumer_key in consumer_secret) zahteva začasne poverilnice. Twitter preveri veljavnost poverilnic in v primeru veljavnosti vrne dva parametra:

- oauth_token,
- oauth_callback, ki je v primeru, da je s poverilnicami vse v redu, nastavljen na true.

Aplikacija s prejetimi začasnimi poverilnicami zgradi URL za prijavo v sistem z uporabo uporabniškega računa Twitter in ga pošlje brskalniku. Brskalnik prejeti URL nato prikaže uporabniku.

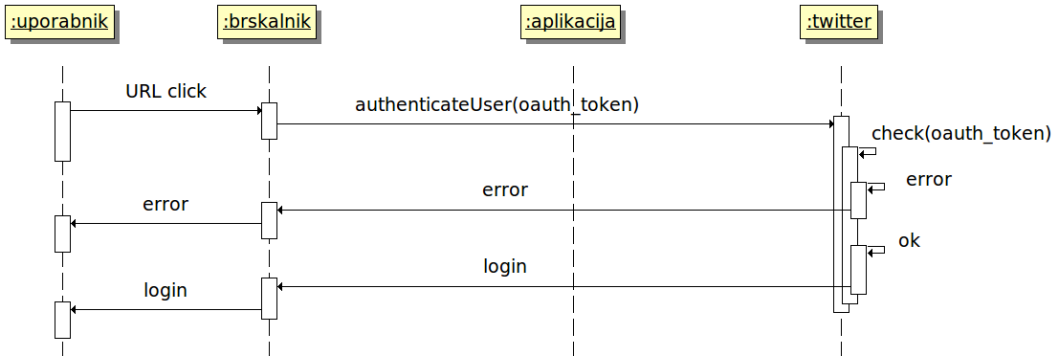


Slika 8.1: Tok dogodkov, ko uporabnik pride na spletno stran.

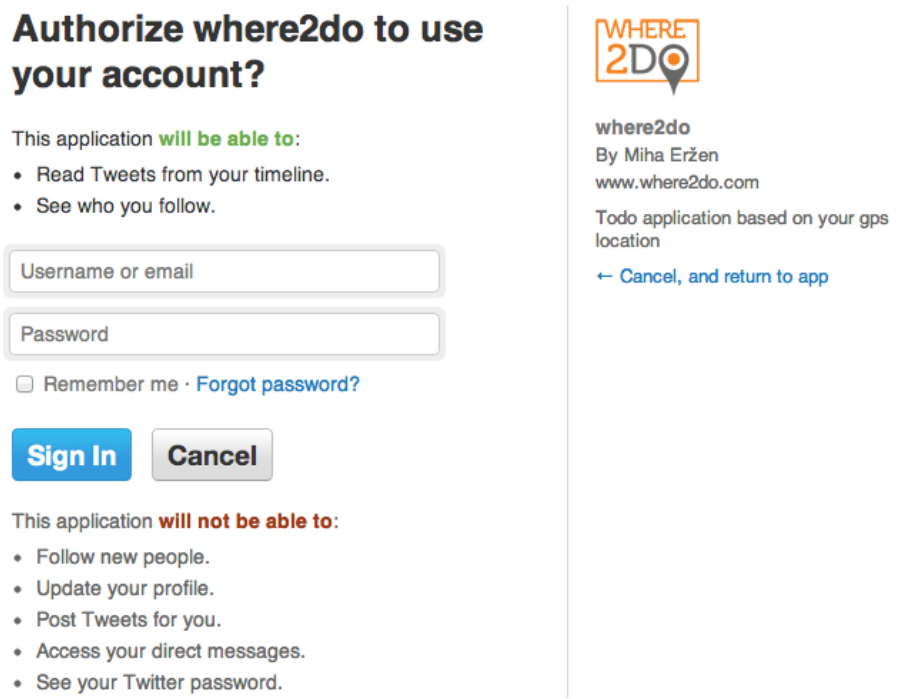
8.2.2 Uporabnik klikne na povezavo

Ko uporabnik klikne na povezavo, ga brskalnik preusmeri na stran Twitter za prijavo. Preko URL-ja se prenese tudi parameter oauth_token, ki ga Twitter preveri.

Če so začasne poverilnice veljavne, se uporabniku prikaže okno za vpis uporabniškega imena in gesla, sicer se mu izpiše napaka o neveljavnosti poverilnic.



Slika 8.2: Tok dogodkov, ko uporabnik klikne na povezavo.



Slika 8.3: Okno Twitter za avtorizacijo aplikacije.

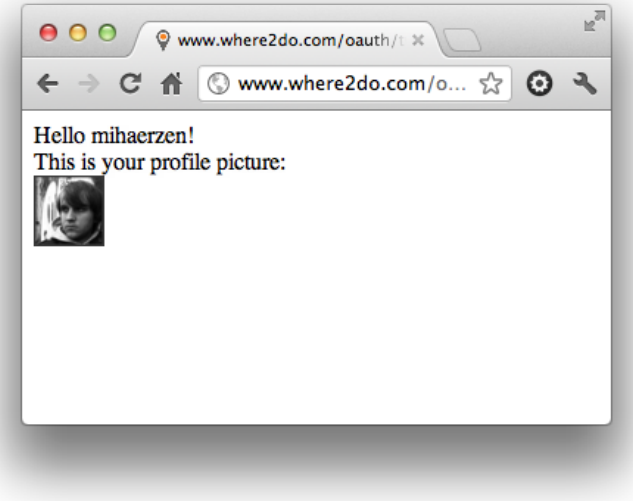
8.2.3 Uporabnik vpiše uporabniško ime in geslo

Uporabnik vpiše uporabniško ime in geslo ter se prijavi v sistem. Če uporabniško ime obstaja in je vnešeno geslo pravo, potem Twitter preusmeri uporabnika nazaj na aplikacijo, in sicer na URL, ki smo ga specificirali ob kreiranju aplikacije (callback URL). Temu URL-ju Twitter doda še dva parametra, in sicer oauth_token in oauth_verifier.

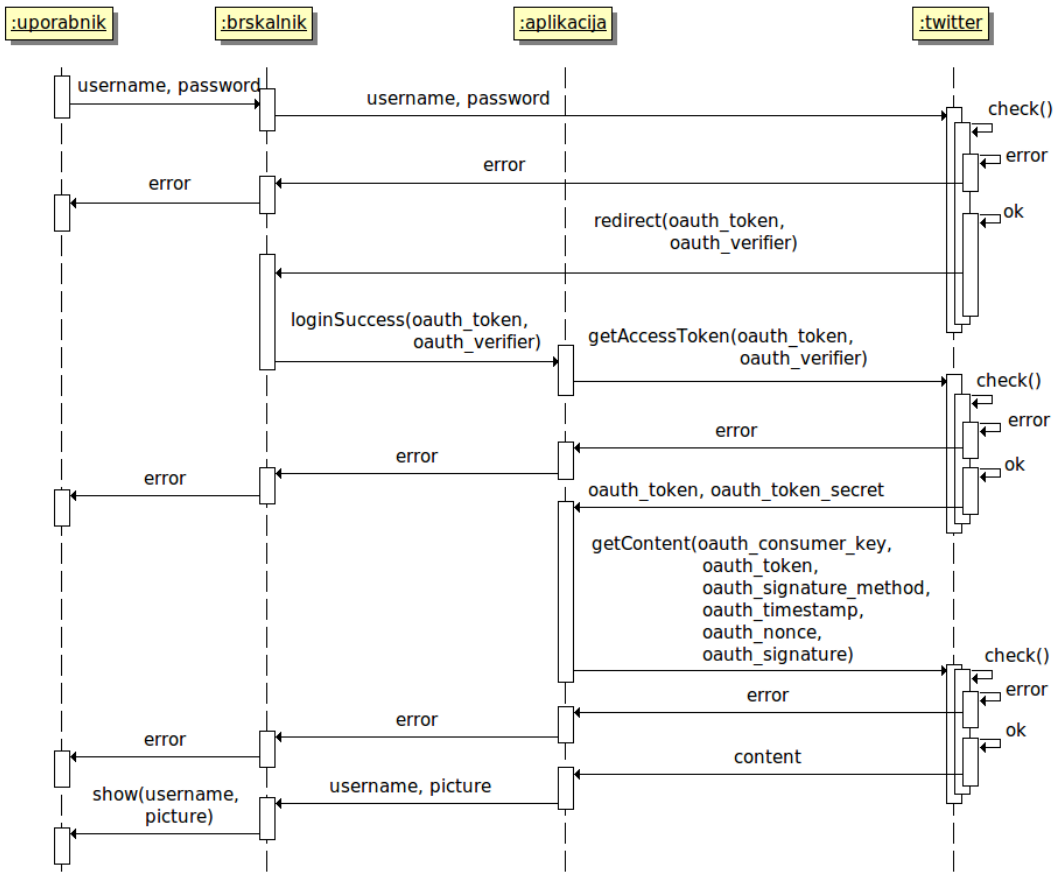
Z dobljenima parametroma aplikacija sedaj od Twitterja zahteva žeton za dostop. Twitter zahtevo potrdi in odgovori s sklopom poverilnic za dostop (oauth_token, oauth_token_secret).

Sedaj lahko od Twitterja pridobimo podatke, ki jih potrebujemo za prijavo uporabnika v aplikacijo. Z metodo POST na Twitter pošljemo zahtevo po uporabnikovih resursih, skupaj s pridobljenimi parametri (oauth_consumer_key, oauth_token, oauth_signature_method, oauth_timestamp, oauth_nonce, oauth_signature).

Strežnik zahtevo potrdi in odgovori z zahtevanimi resursi. Uporabnika prijavimo v sistem ter mu izpišemo pozdravno sporočilo z njegovim uporabniškim imenom in njegovo profilno sliko, ki smo ju pridobili od spletne strani Twitter.



Slika 8.4: Uporabniško ime in profilna slika.



Slika 8.5: Tok dogodkov, ko se uporabnik prijavi.

Poglavje 9

Zaključne ugotovitve

Med pisanjem diplomske naloge sem se seznanil z vsemi funkcionalnostmi odprte avtentikacije. Med iskanjem podobnih protokolov sem ugotovil, da je OAuth trenutno edini odprt generičen protokol, ki omogoča izmenjavo informacij med aplikacijo in uporabnikom brez izmenjave uporabniškega imena in gesla.

Z opcijami in funkcionalnostimi, ki jih ponuja, lahko uporabniku ustvarimo zelo dobro in prijetno uporabniško izkušnjo. Z uporabo OAutha se uporabnik znebi vpisovanja uporabniških imen in gesel, ko brska po internetu, hkrati pa mu omogoča kontrolo nad dostopom aplikacije do njegovih informacij. Uporabnik lahko v vsakem trenutku prekliče dostop aplikacije do njegovih informacij. V tradicionalnem modelu bi uporabnik moral zamenjati uporabniško ime ali geslo.

Menim, da vsak razvijalec, ki v svojo aplikacijo implementira OAuth, pridobi pozitiven odziv uporabnikov. Sam ga bom v prihodnje takrat, ko bo to smiselno, zagotovo uporabil.

Slike

Literatura

[1] (2011) *The OAuth 1.0 Guide*. Dostopno na: `http://hueniverse.com/oauth/guide/`

[2] (2010) *RFC 5849: The OAuth 1.0 Protocol*. Dostopno na: `http://tools.ietf.org/html/rfc5849`

[3] (2012) *Twitter API documentation*. Dostopno na: `https://dev.twitter.com/docs`

[4] (2012) *Authentication - Facebook*. Dostopno na: `http://developers.facebook.com/docs/authentication/`

[5] (2012) *Twitter REST API Resources*. Dostopno na: `https://dev.twitter.com/docs/api`

[6] (2012) *Where2do*. Dostopno na: `http://www.where2do.com`