Facebook Authentication Analysis Using Alloy

Steven Elia and Abe Parvand
CS 259
03/07/2011
Protocol (Client-Side Flow)

User

Content request

Redirect to Facebook, **app_id**, **redirect_uri**, **scope**, **flow_type**

HTTPS OAuth login with **app_id**, **redirect_uri**, **scope**, **flow_type**, Facebook cookie

Checks that **redirect_uri** lies within same domain as application's website

HTTP authorization dialog

Authorize

HTTP 302 redirect to **redirect_uri**#access_token=...

HTTP request **redirect_uri**#access_token=...

Parses **access_token** with client-side code

HTTPS requests on behalf of user, **access_token**
Security Properties

Attacker can't obtain another app's access token

If user authorizes attacker's app, attacker can't force user authorization of a different app

Attacker can't elevate permissions above those explicitly requested by app or granted by user

Assuming SOP enforced correctly, attacker with app in iframe can't force app to make requests
Analysis Method

Alloy

Numerous web and browser interactions
Redirects
Cookies
HTTP(S)

Similar to other protocols being analyzed like OpenID and OAuth
Avoids similar Murphi finite-state machines
Focuses on Facebook-specific issues
// We model Facebook as a principal with its own servers
one sig Facebook extends NormalPrincipal {} {some servers}
one sig FacebookNetworkEndpoint extends HTTPServer{}

fact {
  all fb:Facebook | fb.servers in FacebookNetworkEndpoint
}

// App is a principal with id and secret shared with FB
sig Application extends Principal {
  id:ApplicationId,
  secret:ApplicationSecret
}

// Apps have different DNS and servers than FB
fact ApplicationsNotRunByFacebook {
  all app:Application, fb:Facebook, fbdns:FacebookDNS |
  no (app.dnslabels & fbdns) and
  no (app.servers & fb.servers)
}
// An attacker cannot obtain another app's access token.
// We write this by saying that if there is an access
// token, then there is only one app that got it.
assert AccessTokenBelongsToOneApp {
    all at:AccessToken, app1:Application,
    app2:(Application - app1) |
    AppHasAccessTokenForSomeParameters[app1, at] implies not AppHasAccessTokenForSomeParameters[app2, at]
}
assert PermissionsAreThoseGrantedByUser{
  all at:AccessToken |
  no scope:ScopeQueryPair | {
    // There is some scope in the symmetric difference of
    // those in at and those satisfying AppHasAccessToken
    some ((scope.value.scopeset - at.accessscopes) +
      (at.accessscopes - scope.value.scopeset))
    some flowtype:FlowTypeQueryPair, appid:AppIdQueryPair,
      redirecturi:RedirectUriQueryPair,
      code:CodeQueryPair,
      http_trans:HTTPTransaction | {
      // access token actually the one sent in http_trans
      some atheader:(AccessTokenResponseHeader &
        http_trans.resp.headers) |
        atheader.accesstoken = at
      AppHasAccessToken[at.accessapp, scope, flowtype, appid,
        redirecturi, code, http_trans]
    }
  }
}
Obligatory Alloy Visualization
Two Man in the Middle DOS Attacks

Recall beginning of protocol

Subject to a network attack if
1) App is not hosted on the client
2) User has to initiate with an unencrypted HTTP request
Man in the Middle DOS Attack 1

Decrease Permissions

User ➔ Attacker ➔ Application ➔ Facebook

Content request

Redirect to Facebook, **app_id**, **redirect_uri**, **flow_type**, **scope=email,user_videos**

Redirect to Facebook, **app_id**, **redirect_uri**, **flow_type**, **scope=email**

HTTPS OAuth login with **app_id**, **redirect_uri**, **scope=email**, **flow_type**
Man in the Middle DOS Attack 2

Increase Permissions

User → Attacker
Content request

Redirect to Facebook, `app_id`, `redirect_uri`, `flow_type`, `scope=email,user_videos`

Redirect to Facebook, `app_id`, `redirect_uri`, `flow_type`, `scope=email,user_videos,read_mailbox,xmpp_login,publish_checkins,sms`

HTTPS OAuth login with `app_id`, `redirect_uri`, `scope=...`, `flow_type`
Man in the Middle DOS Attack

Increase Permissions

[Image of a permission request from a website showing various options like Access my basic information, Send me email, Access messages in my inbox, Access Facebook Chat, Send me SMS messages, Check-ins, Access my photos and videos.]
Importance of SOP for Client-Side Flow

If SOP (for DOM) is not correctly enforced, attacker in parent iframe could get access token. This was verified by our model.

// If one iframe gets an access token, the other should not be // able to read it.
assert NoCrossDomainIFrameCanObtainAnothersAccessToken {
    no disj ifa, ifb: IFrameContext |
    !FramesAreSameOrigin[ifa, ifb] and
    FrameCanReadOthersAccessToken[ifa, ifb]
}

// Return true if iframe b (ifb) can obtain an access token and // ifa is able to read it.
pred FrameCanReadOthersAccessToken[ifa: IFrameContext, ifb: IFrameContext] {
    FrameCanReadOtherFrameLocation[ifa, ifb] and
    IFrameCanObtainAnAccessToken[ifb]
}
Initial security properties hold in web attacker threat model

Permission integrity fails in network attacker threat model

Recommend that all remotely hosted applications serve Facebook Authentication pages over HTTPS

Client-Side Flow is tightly coupled with correct SOP DOM access enforcement in the browser. It follows that one loophole in the latter compromises the former.