International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 2 Issue 11, November - 2013

Elevating the performance of social sign-on protocol OAuth 2.0

Gurleen, Research Scholar, BBSBEC,India Aggarwal, D. Professor, BBSBEC, India

Abstract—**Over the past few decades, social networking has been a crucial part of every individual's life, which has led to social sign-on concept. The elementary objective of Social sign-on protocol is to provide succor to the user. It is a form of single sign-on using existing login information from a social networking service such as Facebook, Twitter or Google+ to sign into a third party website in lieu of creating a new login account specifically for that website. OAuth 2.0 is an open authorization protocol which enables applications to access each other's data by social sign on. In this paper, we have discussed the hurdles for wide adoption of this protocol and also our model for elevating the performance of this Authentication protocol.**

Index Terms—**Authentication, Networking, OAuth, Protocol, Social sign-on, Third party login**

I. Introduction

OAuth 2.0 is an open authorization protocol which enables applications to access each other's data by social sign on.[3] OAuth 2.0 is the next evolution of the OAuth protocol which was developed in late 2006. OAuth 2.0 focuses on client developer simplicity while providing specific authorization flows for web applications, desktop applications, mobile phones, and living room devices. It is an authorization protocol for third part login[7].

II. Overview of OAuth 2.0

A. Roles

OAuth defines four roles:

1.Resource owner:used to accord access to a protected resource.

2.Resource server: hosts the protected resources, which accepts and responds to protected resource requests using access tokens.

3.Client:is the entity which makes protected resource requests on behalf of the resource owner and with its authorization. There are two types of clients: Confidential and Public.

4.Authorization server:issues access tokens to the client after successfully authenticating the resource owner and obtaining authorization.

B. Authorization Grant

An authorization grant is a credential representing the resource owner's authorization used by the client to obtain an access token. There are four grant types:

1. Authorization Code:It is obtained by using an authorization server as an intermediary between the client and resource owner. [5]
2. Implicit: In the implicit flow, the client is issued an access token directlyinstead of issuing the client an authorization code. The grant type is implicit as no intermediate credentials are issued.[5]
3. Resource Owner Password Credentials:The resource owner password credentials (i.e. username and password) can be used directly as an authorization grant to obtain an access token.[5]
4. Client Credentials: The client credentials can be used as an authorization grant when the authorization scope is limited to the protected resources under the control of the client, or to protected resources previously arranged with the authorization server.[5]

III. Workflow of OAuth 2.0

As shown in Fig 1, firstly the client requests authorization from the resource owner. The authorization request can be made directly to the resource owner (as shown), or preferably indirectly via the authorization server as an intermediary. The client receives an authorization grant. The client requests an access token by authenticating with the authorization server and presenting the authorization grant. The authorization server authenticates the client and validates the authorization grant, and if valid issues an access token. The client requests the protected resource from the resource server and authenticates by presenting the access token. The resource server validates the access token, and if valid, serves the request.

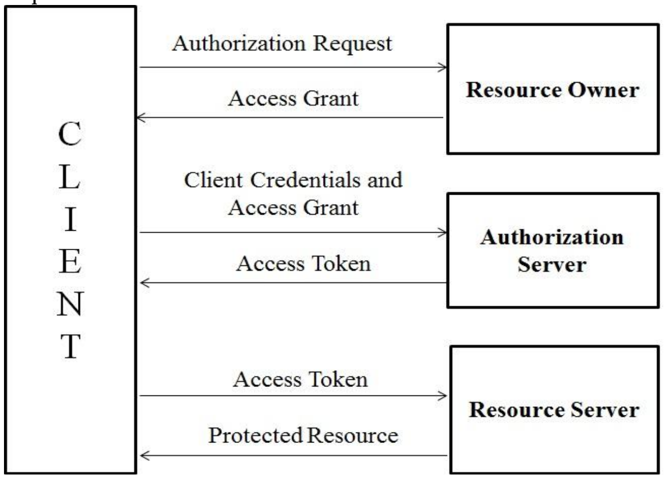


Fig 1 Workflow of the protocol

IV. Hurdles Faced by OAuth 2.0

However, with the emergence of new OAuth Protocol 2.0, simplicity and performance increased, but also many limitations of OAuth 2.0 came into existence:

• CSRF Attack - OAuth2.0 is prone to CSRF (cross-site request forgery) attack. It’s also known as session riding or XSRF.Cross site request forgery is the type of attack when an attacker forces victim's web browser to perform an unwanted action on a user trusted website without user's interaction in this action.[1] This attack exploits the trust of a website on the user's browser. In a recent CSRF attack against residential ADSL routersin Mexico, an e-mail with a malicious IMG tag was sent to victims. By viewingthe email message, the user initiated an HTTP request, which sent a routercommand to change the DNS entry of a leading Mexican bank, making any subsequent access by a user to the bank go through the attacker’s server.[7]

• Unbounded tokens - In 1.0, the client has to present two sets of credentials on each protected resource request, the token credentials and the client credentials. In 2.0, the client credentials are no longer used. This means that tokens are no longer bound to any particular client type or instance. [6]

• Bearer tokens - 2.0 got rid of all signatures and cryptography at the protocol level. Instead it relies solely on TLS (Transport Layer Security).[6]

• Expiring tokens - 2.0 tokens can expire and must be refreshed. This is the most significant change for client developers from 1.0 as they now need to implement token state management. The reason for token expiration is to accommodate self-encoded tokens – encrypted tokens which can be authenticated by the server without a database look-up. Because such tokens are self-encoded, they cannot be revoked and therefore must be short-lived to reduce their exposure. Whatever is gained from the removal of the signature is lost twice in the introduction of the token state management requirement.[6]

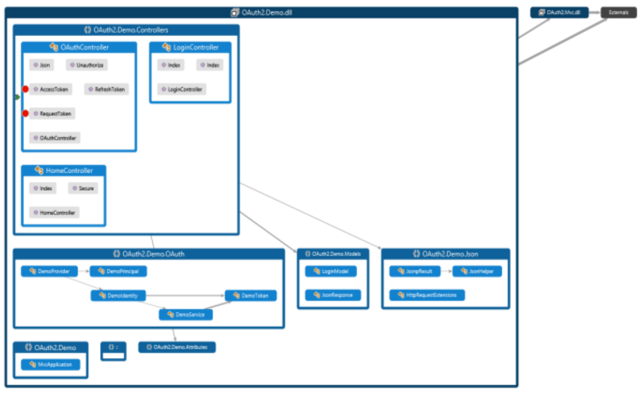
• Grant types - In 2.0, authorization grants are exchanged for access tokens. Grant is an abstract concept representing the end-user approval. It can be a code received after the user clicks ‘Approve’ on an access request, or the user’s actual username and password. The original idea behind grants was to enable multiple flows. 1.0 provides a single flow which aims to accommodate multiple client types. 2.0 adds significant amount of specialization for different client type.[6]

V. Model for Enhancing the Performance

We have developed an application to overcome the hurdles and to enhance the performance of this protocol.We have discussed its basic algorithm and displayed its results.

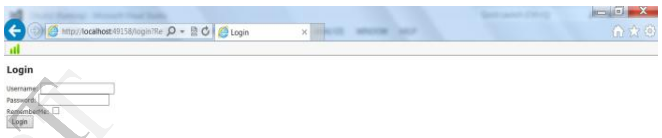
A. Outline and Architecture

Cross-site request forgery, also known as one-click attack or session riding andabbreviated as CSRF or XSRF, is an attack against web applications.In a CSRF attack, a malicious web page instructs a victim user’s browser tosend a request to a target website. It occurs when a malicious web site, email or web
forum causes a victim’s web browser to perform an undesired action on a trusted web site.[2] If the victim user is currently logged into the
target website, the browser will append authentication tokens such as cookies tothe request, authenticating the maliciousrequest as if it is issued by the user.



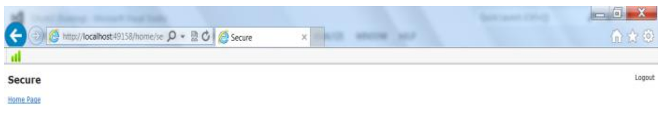
B. Results

When the code is executed, a valid access token is generated after checking the credentials and then the authentication is completed by showing the message that it is a secure resource.



Login
Username:
Password:
Remember Me:
Login

Fig 2. Verification of Credentials



Secure
Logout
Home Page

Fig 3. Conformation page for secure resource access

VI. Conclusion

OAuth 2.0 presents an exalted concept of social sign-on by providing more security than traditional concepts. But, it can be more secure if all the confrontation faced by this protocol is met.

Appendix

// Code for Login Module

```
public class LoginController : Controller

[HttpGet]
public ActionResult Index(string returnUrl)
{
var response = OAuthServiceBase.Instance.RequestToken();
 return View(new LoginModel
  {
            RequestToken = response.RequestToken,
            ReturnUrl = returnUrl
  });


[HttpPost]
public ActionResult Index(string requestToken, string
username, string password, bool? rememberMe, string
returnUrl)
{
varaccessResponse =
OAuthServiceBase.Instance.AccessToken(requestToken,
"User", username, password,
rememberMe.HasValue&&rememberMe.Value);
Session["UserAuthenticated"] = accessResponse;
if (!accessResponse.Success)
{
OAuthServiceBase.Instance.UnauthorizeToken(requestToken)
varrequestResponse =
OAuthServiceBase.Instance.RequestToken();
return View(new LoginModel
{
RequestToken = requestResponse.RequestToken,
 Username = username,
RememberMe =
rememberMe.HasValue&&rememberMe.Value,
ErrorMessage = "Invalid Credentials",
 ReturnUrl = returnUrl
 });
 }
ViewData["ReturnUrl"] =
String.IsNullOrEmpty(returnUrl)? "/": returnUrl;
return View("Success", accessResponse);
}}}
```

// Code for Authorization

```
namespace OAuth2.Demo.Controllers
{
 public class HomeController : Controller
{
 public ActionResult Index()
{
return View();
}
[Authorize]
public ActionResult Secure()
{
if (Session["UserAuthenticated"] != null)
{
return View();
}
else
{

return View("index.cshtml");
}}}}
```

// Code for Request token, Access Token, Refresh Token

```
namespace OAuth2.Demo.Controllers
{
public class OAuthController : Controller
{
[AcceptVerbs(HttpVerbs.Get | HttpVerbs.Post)]
public ActionResult RequestToken()
{
var response = OAuthServiceBase.Instance.RequestToken();
return Json(response, JsonRequestBehavior.AllowGet);
}
[AcceptVerbs(HttpVerbs.Get | HttpVerbs.Post)]
public ActionResultAccessToken(string grant_type, string
username, string password, bool? persistent)
{
varrequestToken = Request.GetToken();
var response =
OAuthServiceBase.Instance.AccessToken(requestToken,
grant_type, username, password,
persistent.HasValue&&persistent.Value);
return Json(response, JsonRequestBehavior.AllowGet); }
[AcceptVerbs(HttpVerbs.Get | HttpVerbs.Post)]
public ActionResultRefreshToken(string refreshToken)
{
 if (String.IsNullOrEmpty(refreshToken))
refreshToken = Request.GetToken();
var response =
OAuthServiceBase.Instance.RefreshToken(refreshToken);
return Json(response, JsonRequestBehavior.AllowGet);
}
Authorize]
[AcceptVerbs(HttpVerbs.Get | HttpVerbs.Post)]
public ActionResultUnauthorize()
{
var response = new JsonResponse();
varaccessToken = Request.GetToken();
response.Success =
OAuthServiceBase.Instance.UnauthorizeToken(accessToken);
return Json(response, JsonRequestBehavior.AllowGet);
}}}
```

ACKNOWLEDGMENT

" **Gur bin ghorandhaar, Guru bin samajnaaavai**"

Waheguru, The Almighty is to be thanked first and foremost for everything that has been destined for me. I am grateful that my destiny bestowed upon me Mr. Deepak Aggarwal as Guru and guide for my research work. I am indebted to Mr. Deepak Aggarwal for supervising my work with his outstanding knowledge and countless time. I would express my gratitude to Mr. R S Uppal for allowing me to draw upon his precious

and valuable time. I am also grateful to all my family members for their immense love..

REFERENCES

1. Siddiqi, M. , Verma, D. ," Cross Site Request Forgery: A common web application weakness", Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on 27-29 May 2011, pp 538-543.

2. Boyan Chen, Zavarsky, P., Ruhl, R. , Lindskog, D., "A Study of the Effectiveness of CSRF Guard", 2011 ieee third international conference on social computing ,9-11 Oct. 2011,pp 1269-1272.

3. Barry Leiba, "OAuth Web Authorization Protocol ", www.computer.org/internet computing, Vol. 16, No. 1. January/February, 2012

4. M. Noureddine, " A provisioning model towards OAuth 2.0 performance optimization", Cybernetic Intelligent Systems (CIS), 2011 IEEE 10th International Conference, Sept. 2011, pp. 76-80

5. D. Hardt, http://tools.ietf.org/html/draft-ietf-oauth-v2Eran Hammer, "OAuth 2.0 and the Road to Hell", http://hueniverse.com/2012/07/oauth-2-0-and-the-road-to-hell/ , July 26, 2012.

6. Tatiana, A. , "Cross-Site Request Forgery: Attack and Defense", Consumer Communications and Networking Conference (CCNC), 2010 7th IEEE, 9-12 Jan. 2010, pp 1-2

7. Lin, X.L., Zavarsky, P., Ruhl, R., Lindskog, D., :Threat Modeling for CSRF Attacks. In: the 2009 International Conference on Computational Science and Engineering, Vancouver (2009).

8. Fung B. S. Y., "A Fine-Grained Defense Mechanism Against General Request Forgery Attacks", In *Proc. of IEEE/IFIP DSN Student Forum*, 2011.

9. Alexa the Web Information Company. http://www.alexa.com.