УДК 004.023

PhD Nataliia Bahniuk, Pavlo Mykhailutsa, Andrii Khrystinets.
Lutsk National Technical University

WEB SERVICE AUTHORIZATION FOR LOCAL NETWORK USERS

Багнюк Н.В. к.т.н., Михайлуца П., Христинець А.О. Web авторизація для користувачів локальної мережі. У статті було проаналізовано основні методи аторизації для web. Дані методи використовуються для процесу розробки та побудови логічної архітектури web сайтів. Основною задачею аналізу було знайти шлях безпечної інтеграції користувачів локальної (корпоративної) мережі у web авторизацію, та надати можливість входу з використанням персональних локальних даних на web сайт. Усі методи передбачають лишу структуровані розробки, що базуються на серверній та фронтальній частині, у більшості випадків серверна частина використовується для обслуговування API запитів фронтальної частини.

Багнюк Н.В. к.т.н., Мыхайлуца П., Христинец А.О. Web авторизация пользователей локальной сети. В статье было проанализировано основные методы авторизации для web. Эти методы используются для процесса разработки и построения логической структуры web сайтов. Основной задачей анализа было найти способ безопасной интеграции пользователей локальной (корпоративной) сети в web авторизацию, и предоставить возможность входа с персональными локальными данными на web сайте. Все методы предусматривают только структурированные разработки, которые базируются на серверной и фронтальной частях, в большинстве случаев серверная часть используется для обслуживания API запросов от фронтальной части.

PhD Nataliia Bahniuk, Pavlo Mykhailutsa, Khrystinets A.O. Web service authorization for local network users. In article was analyzed main methods of authorization for web service. These methods are using for web site developing process and architecture logical schema building. Main analysis point is to find the security way to integrate local network (corporate) users to web service authorization, and give a chance to login with personal local credentials into web site. All methods covers only structured builds, which based on backend and frontend, in most cases backend is using to serve API requests from frontend part.

Formulation of scientific problems.

Integration of local network users in to global surface creates a many security requirements. One of the main security issue is to lead all implementation to a few most reliable points: integrity, confidentiality, availability. In order to prior is to analyze all sides of login process for web service and create protected module for implementing local users authorization. Only accurate and complete authorization logic for web services can solve the security problems.

Analysis of research.

Questions about security models for web-based applictions and formulation of how to build web service as application discovered in[2]. In this article scientists shows how to build models in order to modern security requirements.

Second well known work in security science that can be used for authorization service development is and[3] republished few times later in 2005, 2008 and 2011 [4].

The last but not the least is[5].

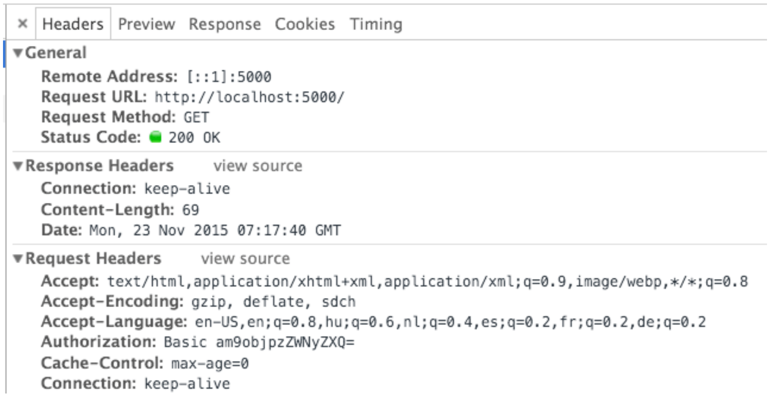
One of the newest article is[6] This invent was published in 2018-01-02 and gained a grant from US government. Good core understanding how to deal with network users and groups are showing in [7]. Same as previous invent gain US government grant and Google patent for future invent [8].

Presentation of the main material and the justification of the results.

There is many type of authentification for web service or sites.

HTTP Basic authentication is a method for the client to provide a username and a password when making a request. This is the simplest possible way to enforce access control as it doesn't require cookies, sessions or anything else. To use this, the client has to send the Authorization header along with every request it makes. The username and password are not encrypted (pic. 1.). There is only one way to up security level of these requests – just to use encrypted connection with SSL/TLS, but if a website uses weak encryption, or an attacker can break it, the usernames and passwords will be exposed immediately. There is no way to log out the user using HTTP Basic authentication.

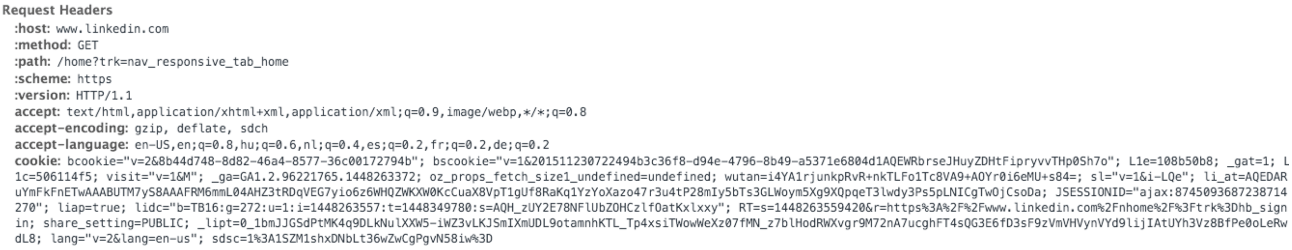
© Багнюк Н.В., Михайлуца П., Христинець А.О



```
× Headers Preview Response Cookies Timing
▼General
   Remote Address: [::1]:5000
   Request URL: http://localhost:5000/
   Request Method: GET
   Status Code: ● 200 OK
▼Response Headers     view source
   Connection: keep-alive
   Content-Length: 69
   Date: Mon, 23 Nov 2015 07:17:40 GMT
▼Request Headers     view source
   Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
   Accept-Encoding: gzip, deflate, sdch
   Accept-Language: en-US,en;q=0.8,hu;q=0.6,nl;q=0.4,es;q=0.2,fr;q=0.2,de;q=0.2
   Authorization: Basic am9objpzZWNyZXQ=
   Cache-Control: max-age=0
   Connection: keep-alive
```

Picture 1. – HTTP request header structure

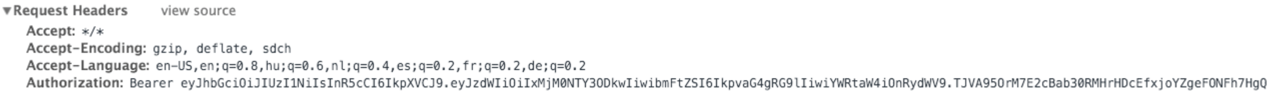
Cookies is another option than HTTP request header to send authorization data (pic. 2.), but if cookies disabled in users browser or project will be high-structured and will use REST based API it might create new overloading on stage of reproduce this requests.



```
Request Headers
  :host: www.linkedin.com
  :method: GET
  :path: /home?trk=nav_responsive_tab_home
  :scheme: https
  :version: HTTP/1.1
  accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
  accept-encoding: gzip, deflate, sdch
  accept-language: en-US,en;q=0.8,hu;q=0.6,nl;q=0.4,es;q=0.2,fr;q=0.2,de;q=0.2
  cookie: bcookie="v=2&8b44d748-8d82-46a4-8577-36c00172794b"; bscookie="v=1&201511230722494b3c36f8-d94e-4796-8b49-a5371e6804d1AQEWRbrseJHuyZDHtFipryvvTHp0Sh7o"; L1e=108b50b8; _gat=1; L
  1c=506114f5; visit="v=1&M"; _ga=GA1.2.96221765.1448263372; oz_props_fetch_size1_undefined=undefined; wutan=i4YA1rjunkpRvR+nkTLFo1Tc8VA9+AOYr0i6eMU+s84=; sl="v=1&i-LQe"; li_at=AQEDAR
  uYmFkFnETwAAABUTM7yS8AAAFRM6mmL04AHZ3tRDqVEG7yio6z6WHQZWKXW0KcCuaX8VpT1gUf8RaKq1YzYoXazo47r3u4tP28mIy5bTs3GLWoym5Xg9XQpqeT3lwdy3Ps5pLNICgTwOjCsoDa; JSESSIONID="ajax:8745093687238714
  270"; liap=true; lidc="b=TB16:g=272:u=1:i=1448263557:t=1448349780:s=AQH_zUY2E78NFlUbZOHCzlfOatKxlxxy"; RT=s=1448263559420&r=https%3A%2F%2Fwww.linkedin.com%2Fnhome%2F%3Ftrk%3Dhb_sign
  in; share_setting=PUBLIC; _lipt=0_1bmJJGSdPtMK4q9DLkNulXXW5-iWZ3vLKJSmIXmUDL9otamnhKTL_Tp4xsiTWowWeXz07fMN_z7blHodRWXvgr9M72nA7ucghFT4sQG3E6fD3sF9zVmVHVynVYd9lijIAtUYh3Vz8BfPe0oLeRw
  dL8; lang="v=2&lang=en-us"; sdsc=1%3A1SZM1shxDNbLt36wZwCgPgvN58iw%3D
```

Picture 2. – HTTP request header structure with cookie based authentication

Tokens is most popular authentication logic for now, 90% of newly created web services or sites based on this type of authentication. Tokens in basics is just a generated long string value. For generation we can use different algorithms, such as HASH or MD5 function, or even more crypto-algorithms with different keys length. For example, one of most popular services for generating token is JWT (JSON Web Token) (pic. 3.). There are many different modules (packages) for different languages already created and free.



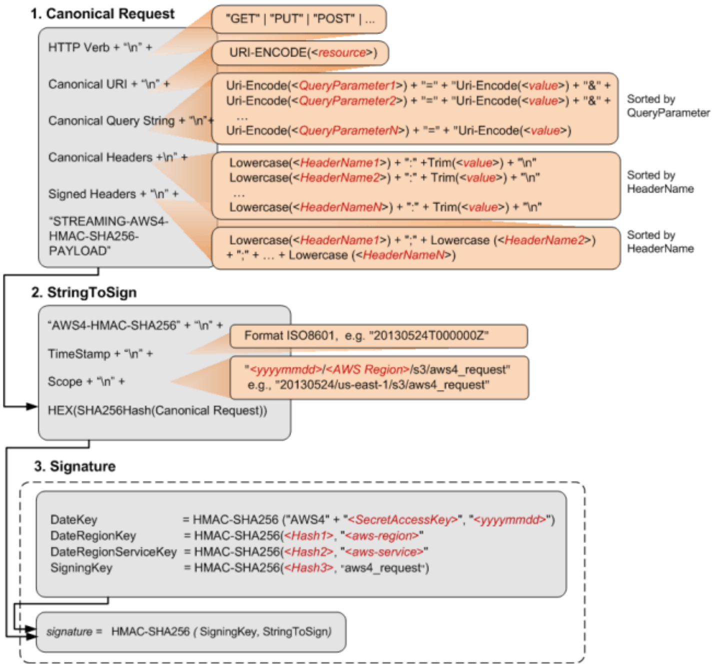
```
▼Request Headers     view source
   Accept: */*
   Accept-Encoding: gzip, deflate, sdch
   Accept-Language: en-US,en;q=0.8,hu;q=0.6,nl;q=0.4,es;q=0.2,fr;q=0.2,de;q=0.2
   Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ
```

Picture 3. – HTTP request header structure with JWT Token based authentication

Signatures. Either using cookies or tokens, if the transport layer for whatever reason gets exposed credentials are easy to access - and with a token or cookie the attacker can act like the real user. A possible way to solve this - at least when we are talking about APIs and not the browser is to sign each request. To make it work, both the consumer of the API and the provider have to have the same private key. Once you have the signature, you have to add it to the request, either in query strings or HTTP headers. Also, a date should be added as well, so you can define an expiration date. But it is realy hard to implement from user side (pic. 4.).

One-Time passwords algorithms generate a one-time password with a shared secret and either the current time or a counter. These methods are used in applications that leverage two-factor authentication: a user enters the username and password then both the server and the client generates a one-time password. The main problem for this type of authentication with the shared-secret (if stolen) user tokens can be emulated.

The problem for local network or corporate network users is to lead the rule of real logins or passwords confidentiality protect. For this kind of implementation we can use redirecting logic, where login and password is not saving on web server (backend or frontend part). In this case login form in web service or on site will directly sends all authorization requests directly to authorization server. Second problem for this kind of structure is to serve high level of integrity. There is no secret that in different organizations and diferent companies are using diferent kind of authorization servers. For example for company with Windows main technology (it means most PCs and server operate by Windows OSs) authentication process will be driven by Active Directory with Kerberos authorization, but for Linux or Unix based networks the leader will be POSIX authorization.


1. Canonical Request
HTTP Verb + "\n" +
Canonical URI + "\n" +
Canonical Query String + "\n" +
Canonical Headers +\n" +
Signed Headers + "\n" +
"STREAMING-AWS4-HMAC-SHA256-PAYLOAD"
"GET" | "PUT" | "POST" | ...
URI-ENCODE(<resource>)
Uri-Encode(<QueryParameter1>) + "=" + "Uri-Encode(<value>) + "&" +
Uri-Encode(<QueryParameter2>) + "=" + "Uri-Encode(<value>) + "&" +
...
Uri-Encode(<QueryParameterN>) + "=" + "Uri-Encode(<value>)
Sorted by QueryParameter
Lowercase(<HeaderName1>) + ":" +Trim(<value>) + "\n"
Lowercase(<HeaderName2>) + ":" + Trim(<value>) + "\n"
...
Lowercase(<HeaderNameN>) + ":" + Trim(<value>) + "\n"
Sorted by HeaderName
Lowercase(<HeaderName1>) + ";" + Lowercase (<HeaderName2>) + ";" + ... + Lowercase (<HeaderNameN>)
Sorted by HeaderName
2. StringToSign
"AWS4-HMAC-SHA256" + "\n" +
TimeStamp + "\n" +
Scope + "\n" +
HEX(SHA256Hash(Canonical Request))
Format ISO8601, e.g. "20130524T000000Z"
"<yyyymmdd>/<AWS Region>/s3/aws4_request"
e.g., "20130524/us-east-1/s3/aws4_request"
3. Signature
DateKey = HMAC-SHA256 ("AWS4" + "<SecretAccessKey>", "<yyyymmdd>")
DateRegionKey = HMAC-SHA256(<Hash1>, "<aws-region>"
DateRegionServiceKey = HMAC-SHA256(<Hash2>, "<aws-service>"
SigningKey = HMAC-SHA256(<Hash3>, "aws4_request")
signature = HMAC-SHA256 (SigningKey, StringToSign)

Picture 4. – Signature based authentication

Solving the problem of cross-platform networks is hiding in implementing server structure what can include Kerberos, POSIX and many others authentication logic. On this stage, the leader will be integrating into the local network LDAP (Lightweight Directory Access Protocol) server.

A client starts an LDAP session by connecting to an LDAP server, called a Directory System Agent (DSA), by default on TCP and UDP port 389, or on port 636 for LDAPS (LDAP over SSL, see below). The client then sends an operation request to the server, and the server sends responses in return. With some exceptions, the client does not need to wait for a response before sending the next request, and the server may send the responses in any order. In addition the server may send "Unsolicited Notifications" that are not responses to any request, e.g. before the connection is timed out.

A common alternative method of securing LDAP communication is using an SSL tunnel. The default port for LDAP over SSL is 636. The use of LDAP over SSL was common in LDAP Version 2 (LDAPv2) but it was never standardized in any formal specification. This usage has been deprecated along with LDAPv2, which was officially retired in 2003. Global Catalog is available by default on ports 3268, and 3269 for LDAPS.

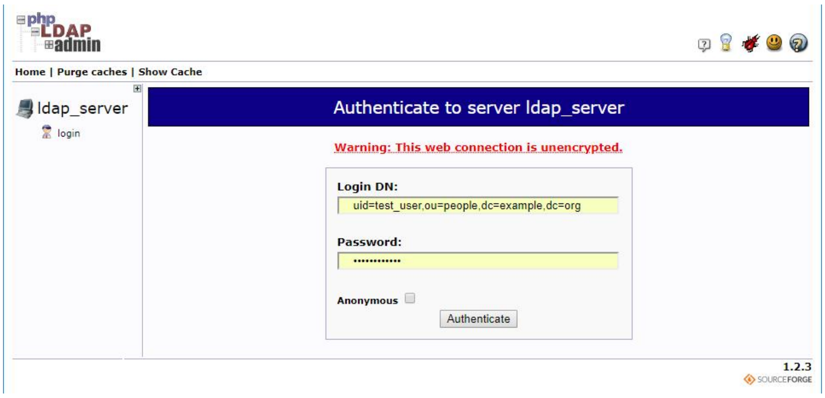
Simplest way to build LDAP server is to create docker-compose with container configuration:

```
version: '2'
services:

  openldap:
    image: osixia/openldap:1.1.11
    container_name: openldap
    environment:
      LDAP_LOG_LEVEL: "256"
      LDAP_ORGANISATION: "Example Inc."
      LDAP_DOMAIN: "example.org"
      LDAP_BASE_DN: "dc=example,dc=org"
      LDAP_ADMIN_PASSWORD: "admin"
      LDAP_CONFIG_PASSWORD: "config"
      LDAP_READONLY_USER: "false"
      #LDAP_READONLY_USER_USERNAME: "readonly"
      #LDAP_READONLY_USER_PASSWORD: "readonly"
      LDAP_RFC2307BIS_SCHEMA: "false"
      LDAP_BACKEND: "hdb"
      LDAP_TLS: "true"
      LDAP_TLS_CRT_FILENAME: "ldap.crt"
      LDAP_TLS_KEY_FILENAME: "ldap.key"
      LDAP_TLS_CA_CRT_FILENAME: "ca.crt"
      LDAP_TLS_ENFORCE: "false"
      LDAP_TLS_CIPHER_SUITE: "SECURE256:-VERS-SSL3.0"
      LDAP_TLS_PROTOCOL_MIN: "3.1"
      LDAP_TLS_VERIFY_CLIENT: "demand"
      LDAP_REPLICATION: "false"
      KEEP_EXISTING_CONFIG: "false"
      LDAP_REMOVE_CONFIG_AFTER_SETUP: "true"
      LDAP_SSL_HELPER_PREFIX: "ldap"
    tty: true
    stdin_open: true
    volumes:
      - /var/lib/ldap
      - /etc/ldap/slapd.d
      - /container/service/slapd/assets/certs/
    ports:
      - "389:389"
      - "636:636"
    domainname: "example.org"
    hostname: "example.org"

  phpldapadmin:
    image: osixia/phpldapadmin:latest
    container_name: phpldapadmin
    environment:
      PHPLDAPADMIN_LDAP_HOSTS: "openldap"
      PHPLDAPADMIN_HTTPS: "false"
    ports:
      - "8080:80"
    depends_on:
      - openldap
```

where openldap service current LDAP container configuration and phpldapadmin web interface (pic. 5) for managing LDAP server structure.



Authenticate to server ldap_server

Warning: This web connection is unencrypted.

Login DN: uid=test_user,ou=people,dc=example,dc=org

Password: ••••••••••••

Anonymous □

Picture 5. – PHP web interface

All user will looks like ldif file with lines:

```
version: 1
dn: uid=billy,ou=people,dc=example,dc=org
changetype: add
uid: billy
cn: billy
sn: 3
objectClass: top
objectClass: posixAccount
objectClass: inetOrgPerson
loginShell: /bin/bash
homeDirectory: /home/billy
uidNumber: 14583102
gidNumber: 14564100
userPassword: {SSHA}j3lBh1Seqe4rqF1+NuWmjhvtAni1JC5A
mail: billy@example.org
gecos: Billy User
```

In order to docker compose configuration to get access to LDAP server web service should redirect request to one of opened ports 389 or 636. For any manipulation with server web service should to initialize connection to server, than make some operations and close connection. For structured platform with backend and frontend parts any server manipulation are running on backend part, frontend part is using only for getting data and transfer as a request to backend.

All manipulations with LDAP initialization, authorization, manipulation requests represents in python code-base with installed package with basic ldap server functions support (python-ldap==3.0.0):

```
>> import ldap
>> l = ldap.initialize('ldap://localhost'))
>> l.protocol_version = ‘3’
>> l.simple_bind_s(‘cn=username,dc=example,dc=org’, ‘password’)
```

where ‘cn=username,dc=example,dc=org’ full user name with it tree position on LDAP server

For automation of any process with python we can use different functions and call them on demand. Successful response of any python-ldap functions will be LDAPObject.

Conclusion and prospects for further research. Any local user with remote access can not be stored on external web-services but we can use python scripts or function in projects (prior backend part) to authenticate local network users. There is no excuses, for low security, and will be a good point, to remember that we can use SSL connection to protect data from stealing. For cryptography issue we can use keys with different length or algorithm. Any new server connection will be protected. Authorization methods presented in this article can provide fast, and protected authentication process for local network users. All represented code was tested on cloud platform and can be used for real web service implementation. LDAP user authorization contains high level of scientific value and can be used for future research.

1. "Mandatory Reporting of Conventional Generation Performance Data" (PDF). Generating Availability Data System. North American Electric Reliability Corporation. July 2011. pp. 7, 17. Retrieved 13 March 2014.
2. “Security models for web-based applications” by James B. D. Joshi, Walid G. Aref, Arif Ghafoor, Eugene H. Spafford and published in: Magazine Communications of the ACM CACM Homepage archive Volume 44 Issue 2, Feb. 2011 Pages 38-44 ACM New York, NY, USA https://dl.acm.org/citation.cfm?id=359224
3. “System And Method For Managing User Authentication And Service Authorization To Achieve Single-Sign-On To Access Multiple Network Interfaces” created by inventors Pei Chia and Hong Cheng for Panasonic Corp in 2004
4. https://patents.google.com/patent/US20080072301A1/en
5. “A survey of Web security” by A.D. Rubin ; D.E. Geer becomes a part of IEEE standards first published in Published in: Computer (Volume: 31, Issue: 9, Sept. 1998) pages: 34 – 41, but republishing is going till nowadays [http://ieeexplore.ieee.org/abstract/document/708448/?reload=true
6. “Methods and systems for providing secure access to a hosted service via a client application” by Anthony J. Yeates, Pavel A. Dournov, Sumeet Updesh Shrivastava, Shankar Arunachalam Bharadwaj, Donna L. Whitlock [https://patents.google.com/patent/US9858562B2/en
7. “Methods and systems for creating and managing network groups” by Thomas M. Kludy, Ashish Gujarathi, Ricardo Fernando Feijoo
8. https://patents.google.com/patent/US9906461B2/en

© Багнюк Н.В., Михайлуца П., Христинець А.О