Case Study:
LDAP Authentication and Authorization for Open Source Web Applications

Justin B. Alcorn
9/29/2003

SANS Security Essentials
GSEC Practical Assignment
Version 1.4b – Option 2

Abstract

Many open source web applications use relational databases to hold identifying user information. This model works well for stand-alone Internet applications where the user population is self-chosen and is willing to sign up for and maintain an independent identity for the application. When writing applications for a corporate environment, however, the user population is no longer self-selected. I had some specific challenges that I wanted to solve including an IT Security intranet, tracking of audit issues, and monitoring of corporate email systems where open-source tools could be used to quickly and easily put the information into a HTML format. The information, however, had various levels of classification, and users had to be authenticated and authorized to see the specific applications.

I found a significant number of different examples of using a back-end database to hold the authentication information. However, this simply exacerbated an already existing problem – namely, that corporate environments often have many disparate systems, all with different authentication information. A centralized directory service, such as LDAP, is used by many corporations to unify authentication and authorization and to gain administrative control of provisioning. As part of my job, I am involved in unifying all these different authentication mechanisms and building a central authentication and authorization directory. In this paper, I describe the method I used to solve this problem for the open source PHP applications. I present the actual code and a directory model that can be extended for use as a central coporate directory. I implemented the code on my home intranet as a prototype, and that code is the code presented here.

The Problem - Before

High end Open Source web applications, such as Content Management or Groupware systems, depend upon user authentication to handle security. Most have moved to a session-based model rather than using the web server security available in HTTP Basic and Digest authentication. A form is presented to the user, and the posted information is compared to the user information stored to determine access rights.

Most of these Open Source applications store the user authentication in a relational database, usually because the database is used for other purposes. Each application includes installation of user tables with the attributes needed for that application. Installation of another application will install another set of tables. Users will need to be set up in both applications, and maintained separately. Examples of these applications include geeklog (http://www.geeklog.net) , PostNuke (http://www.postnuke.org), typo3 (http://typo3.com) and Mambo Open Source (http://www.mamboserver.com) . Each of these applications includes administrative tools and unique tables for maintaining users. Although useful in a isolated context, these tools do not fit in an environment where many different applications are running that need common authentication credentials.

I also built some specific applications for my own use, both on my home network and for my corporate responsibilities. Home applications included photo galleries for organizing pictures of the family, online address books, and journals of my children's activites. For work I built a security issues tracking database, a database for monitoring the logs on our email content filters, web sites that monitored various network devices using tools such as MRTG (http://www.mrtg.com) and Nagios (http://www.nagios.org). I also had some straight HTML pages that I wanted to have varying levels of classification. Giving people various levels of access to each of these applications meant maintaining user information, and although I wrote many of the applications to take advantage of specific user tables, I still wasn't taking advantage of any central provisioning available in the corporate infrastructure. The default answer was to use Microsoft IIS servers and HTTP authentication to specific directories, allowing the users to use their Windows Network logons to access information. The addition of each Windows Server/SQL Server application increased costs to the enterprise, and the "locking in" of one technology limited our ability to creatively solve problems. I needed to find a way to authenticate users and classify information while still using the open source tools that gave me the flexibility to creatively solve problems.

What I needed was a standards-based way to authenticate users and classify data that would work with any technology.

Putting It Together - During

One of my central responsibilities at my job is to begin to solve this issue of "silos of authentication". Windows networks, email systems, ERP systems, Unix accounts, Web applications, HR systems, and one-off applications are scattered around any enterprise, and provisioning and maintaining these accounts takes a significant amount of effort and resources.

Many of these systems are, to one degree or another, moving to support LDAP authentication mechanisms. A decision had already been made based upon some committments made and software purchased that a corporate LDAP directory would be used. The specific software used for this directory, however, was in the decision process. I decided to go ahead and prototype the authentication mechanisms I would need using freely-available tools, and make sure that I would be able to use any of the commercial directories as needed when final implementation was done.

Decision - Why Sessions?

Apache, the most popular web server in use today[1], has a module available for LDAP authentication of HTTP connections. This implementation, however, did not meet my needs because I require session based authentication. Some of the reasons for using session based authentication instead of HTTP based authentication include the following:

- Session Variables – Sessions can store information about the current state of the transactions and retrieve that information between submissions. Session variables allow stateful applications to be built on the stateless HTTP protocol.
- Session Control – Sessions can be destroyed as well as created, allowing a user to actually log out of an application. HTTP authentication is persistent until the browser is closed. In addition, cookies can be timed out so that idle sessions are automatically logged off.
- Encryption – Although HTTP authentication has a hashed Digest option available[2], most HTTP authenticated sites use only basic authentication. Sites mainly choose this limited method for ease of use, but many assume a common misconception that because the password field is obscured, the password must be encrypted in transit. This is incorrect – the password field in Basic authentication is sent in the clear. However, for session based authentication, if the server is SSL encrypted and the form is submitted with a https:// URL, the authentication credentials are also encrypted.

Decision - Why Directories?

A directory server is essentially an information store that is optimized for reads. The Domain Name System (DNS) is a type of directory server that is optimized for both reads and for the type of information it maintains. A generalized directory server, on the other hand, can store information about people, locations, groups, applications, network devices, and anything else for which there is a need for fast retrieval of information that does not change very often. Directory servers should be stable, extensible and distributed, to allow for redundancy and local administration of the information. The DNS system is fast, stable and distributed, but is not very extensible, and so is not a general directory server.

Directory servers are often seen as lightweight databases, and there is some misunderstanding of the use of directory servers. Although a directory server may be implemented with a database on the back-end, a directory server would

[1] Netcraft Survey, July 2003 © Netcraft, Ltd. http://news.netcraft.com/archives/web_server_survey.html
[2] HTTP Authentication: Basic and Digest Access Authentication http://www.ietf.org/rfc/rfc2617.txt

be a very bad choice for any system that requires any kind of transactional capability. As a matter of fact, directory servers are usually set up to limit updates to a few specific roles, and possibly to allow people to update their own information.

Directory servers allow large organizations to delegate administration of users, networks, names and devices to local administrators, while maintaining a single way of accessing that information from anywhere. Directory servers provide "white pages" functionality, and can bring information in from disparate sources for ease of access, for example updating a person's name, address, and employee information from an HR system while taking their office assignment from an administrative database and the phone number from the PBX system. A network-focused directory server can provide benefits to an enterprise by:

- Facilitating distributed networking by bringing in information from different network resources into a central location. Directory servers are highly scalable and integrate disparate systems using Internet standards.
- Easing network administration by providing a central store of information for provisioning systems.
- Enhancing network reliability and performance by providing a distributed, redundant core of information.
- Unifying access to those distributed resources, with the ability to use aliases to locate network resources which may move locations of even NOS connections, single network logons across sytems, and centralized naming standards.
- Improving security by providing distributed provisioning with central control and providing directory object security through a unified standard.[3]

Decision - Why LDAP?

LDAPv3, the core of which is defined in RFCs 2251 through 2256, is a "lightweight' implementation of the X.500 Directory Services standard. X.500 is considered "heavyweight" because it requires implementations to be written using the seven layer OSI model. LDAP, on the other hand, is implemented using the TCP/IP stack, which is considerably easier to use. There is nothing "lightweight" about the directories themselves – a LDAP implementation can use the same heavy-duty directory data stores used by X.500 implementations. "Lightweight" refers to the ease of access through the TCP/IP protocols.

LDAP is being adopted as the defacto standard for directory access by many organizations and applications. Microsoft's Active Directory, Lotus' Domino Server, Sun/Netscape, Novell, Computer Associates, IBM and many others offer

[3] Sheresh and Sheresh, Undertanding Directory Services, pp. 8-15

implementations of LDAP that differ widely in administration, implementation and data storage, but all can be accessed via an LDAP client and all implement the concept of a directory schema.

Classification Of Data

An important decision in the implementation of any data security scheme is the classification of the data – how will the authorization work? We are moving to a Role-Based Security model, and the design of the LDAP authentication supports that model. In a Role-based model, data is classified according to the roles of the users who will have access to the data. Roles are implemented by including users in groups set up to define each role. So, for example, we have a 'anonymous' group that defines the role of someone whose identity is currently unknown to the application, and an 'allusers' role which is composed of anyone who has a valid logon and is currently logged on. Note that these are both different from the 'all' role, which is everyone, whether logged on or not.

Other roles/groups can be defined by who you are – a 'family' role on a home intranet, a 'it security' group – or a function of your job, for example a 'mail admins' role. Again, your role is defined by including you in the appropriate groups. Data is then classified and secured by making it an 'app' in the LDAP directory. As an example, this paper is classified for authenticated users by including the following entry in the directory:

```
# gsecpaper, app, jalcorn.net
dn: cn=gsecpaper,ou=app,o=jalcorn.net
cn: gsecpaper
member: cn=allusers,ou=group,o=jalcorn.net
labeledURI: http://jalcorn.net/gsecpaper/index.php
~gsecpaper/index.php~~GSEC
 Paper Online~
objectClass: groupofnames
objectClass: labeleduriobject
objectClass: top
```

(Note that there is some extra code in the labeledURI attribute used to modify how the menu is displayed). The 'gsecpaper' application will now be available to any authenticated user of the website, but an authenticated user will be denied access.   The data is then secured by changing the HTML file to a PHP file and including the following code at the beginning:

```
<?php
$appname='gsecpaper';
$needauth=1;
require_once "secure.php";
?>
```

Implementation

I used an implementation example of Session-based authentication as a starting point found at http://martin.f2o.org/php/login. This implementation focused on securing sessions and used the PHP PEAR Database libraries available at http://pear.php.net. Although the PEAR Database module contains a LDAP implementation, I found that the code was poorly documented and difficult to use, so I used the direct LDAP PHP calls documented at http://php.net.

System Software

For the example implementation, I have used the following software:

- OpenLDAP v 2.1.22
- BerkeleyDB v 4.1.25 (Required for OpenLDAP)
- Apache v 1.3.27
- PHP v 4.3.2 (compiled –with-ldap)
- RedHat Linux 8

All software (except the operating system) was compiled from source from the latest releases. Examples are written in PHP, but could also have been written in other languages such as C/C++ or Perl.

LDAP configuration

One of the security concerns with using a relational database to authenticate users is that the application must store and pass its credentials for accessing the database. LDAP on the other hand allows anonymous binding for accessing the directory, allowing us to implement the authentication routines without the need for any password storing. However, the access controls must be set correctly to allow searching the directory data. The access controls set for this example are:

```
access to dn.base="cn=Subschema" by * read
access to attrs=userPassword
        by self write
        by * auth
access to attrs=uid,entry
        by * read
access to dn.children="ou=app,o=jalcorn.net" by * read
access to dn.children="ou=group,o=jalcorn.net" by * read
access to *
        by self write
        by users read
        by anonymous auth
```

This access list allows authentication to the LDAP server with an anonymous bind, and allows the anonymous user to search the 'app' and 'group' organizational units, plus search the person ou and see the uids. Complete LDAP configuration information is beyond the scope of this paper but can be found at the OpenLDAP page (http://www.openldap.com) and more LDAP information can be found at the LDAP Zone (http://www.ldapzone.com) .

Directory Design

To accomplish our goals of authentication and authorization using LDAP directory services, we are going to store three kinds of information in the directory. First, user information including a unique ID and password, along with name and other information. Authorized users will be stored in the organizational unit ou=person and will use the inetOrgPerson object class defined in RFC 2798[4]. The DN for users will be constructed from the uid, rather than the common name (cn), to ensure uniqueness and make logons easier.

Application information will be stored in ou=app. Application objects will be contructed of objectClass groupOfNames (RFC 2256) [5]and labeledURIObject (RFC 2079)[6]. The groupOfNames gives us a 'member' attribute which is a DN of another object (in this case, groups) while the labeledURI attribute will contain a URL and some text describing the application.

Group information will be used to tie the applications and users together, and will be stored in ou=group. Groups are build from the objectClass groupofNames and the 'member' attributes are the DNs of users or of other groups.

[4] Definition of the inetOrgPerson LDAP Object Class http://www.ietf.org/rfc/rfc2798.txt
[5] A Summary of the X.500(96) User Schema for use with LDAPv3 http://www.ietf.org/rfc/rfc2256.txt
[6] Definition of an X.500 Attribute Type and an Object Class to Hold Uniform Resource Identifiers (URIs) http://www.ietf.org/rfc/rfc2079.txt

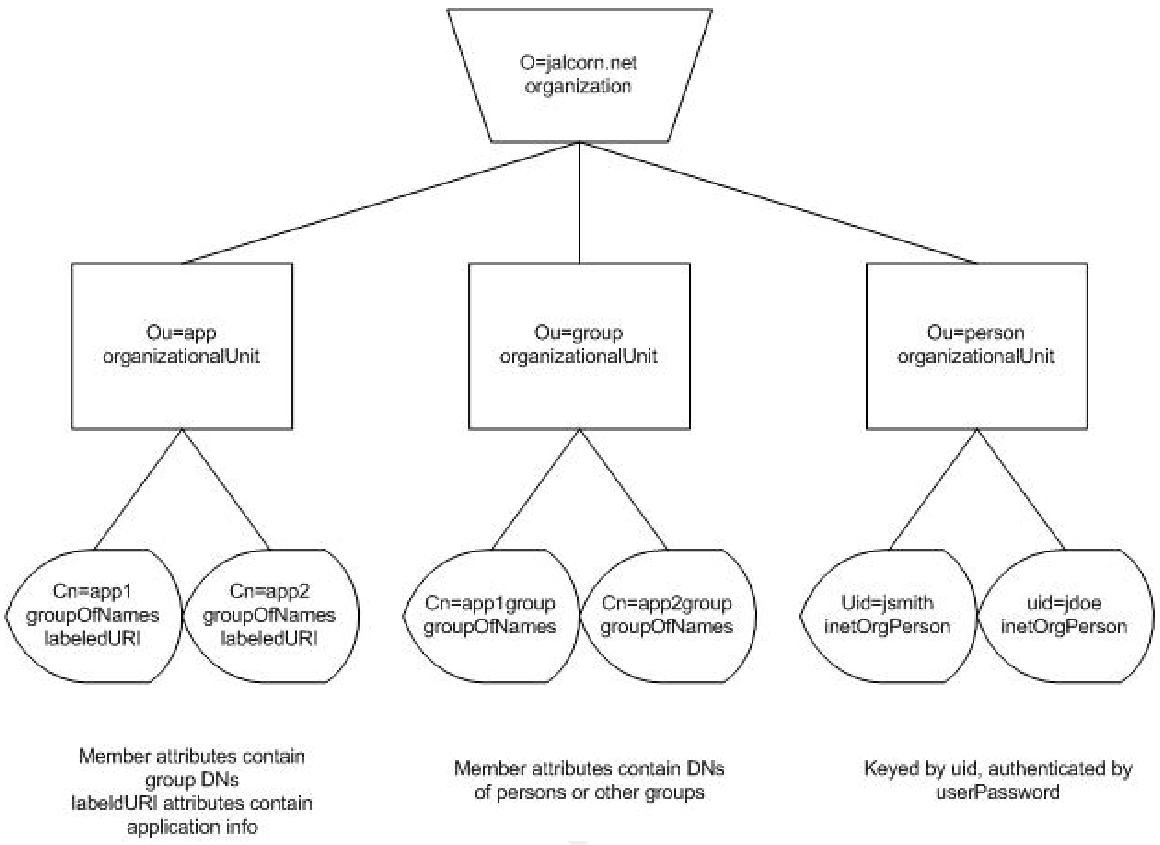


Figure 1 - LDAP Organization

Object Code

The code to implement LDAP integration is contained in 2 PHP classes. The ldapDB is a class representing a LDAP directory connection. The class file ldapDB.php is stored on the web server in a directory specified in the PHP search path. [7]

```
<?php
class ldapDB {
//
// ldapUser.php (c) 2003 Justin B. Alcorn
   /*
   *   This script is  free software;
   *   you can redistribute it and/or modify
   *   it under the terms of the GNU General
   *   Public License as published by
   *   the Free Software Foundation;
   *   either version 2 of the License, or
   *  (at your option) any later version.
   *
   *  The GNU General Public License can be found at
   *  http://www.gnu.org/copyleft/gpl.html.
```

[7] For information on configuring PHP, see http://www.php.net

```
*
*  This script is distributed in the hope
*  that it will be useful,
*  but WITHOUT ANY WARRANTY; without even
*  the implied warranty of
*  MERCHANTABILITY or FITNESS FOR A PARTICULAR
*  PURPOSE.  See the
*  GNU General Public License for more details.
*/
//
// Based on an idea from http://martin.f2o.org/php/login
//
   var $conn = null; // LDAP Connection Object
   var $bound = 0;
   var $anon = 0; // Anonymous Connection
   var $binddn = null; //
   // basedn for all searches
   var $base = null;
   var $host = null;

   function ldapDB($host,$base,$uid="", $pwd="") {
      $this->host = $host;
      $this->base = $base;
      $this->conn = ldap_connect($host) or die("Cannot connect to LDAP $host\n");
      // We always start with an anonymous bind
      ldap_bind($this->conn) or die("Anonymous bind failed\n");
      $this->bound = 1;
      $this->anon = 1;
      $this->binddn = null;
      if ($uid) {
         // Make sure that attempts with NULL passwords do not succeed
         if (! $pwd) {
            $pwd = crypt(microtime());
         }
         $filter = "(cn=$uid*|uid=$uid*)";
         $res = ldap_search($this->conn,$this->base,$filter);
         $cnt = ldap_count_entries($this->conn,$res);
         if ($cnt == 1) {
            $entry = ldap_first_entry($this->conn,$res);
            if ($entry) {
               $bdn = ldap_get_dn($this->conn,$entry);
               if ($bdn) {
                  if (@ldap_bind($this->conn, $bdn, $pwd)) {
                     $this->anon = 0;
                     $this->binddn = $bdn;
                  }
               }
            }
         } else {
            if ($cnt == 0) {
            } else if ($cnt > 1) {
            } else {
            }
         }
      }
   } // function ldapDB

   function isConn() {
      if ($this->conn) {
         return TRUE;
      }
      return FALSE;
   }

   function isBound() {
      if ($this->bound) {
         return TRUE;
      }
      return FALSE;
   }

   function isAuthUser() {
      if ($this->binddn) {
         return TRUE;
      }
      return FALSE;
   }

   function isAnon() {
      if ($this->anon) {
         return TRUE;
      }
      return FALSE;
   }
} // class ldapDB
```

The class' constructor takes the hostname and a basedn as required arguements. It first attempts an anonymous bind to the server, and searches for the user's UID. This allows the directory to be set up hierarchically - the user doesn't need to authenticate with their complete Distinguished Name (i.e. uid=jalcorn,ou=itsecurity,ou=person,o=jalcorn.net) but only needs to use their Unique ID (jalcorn).[8] Note that we allow the uid to be a 'cn' also, this is to allow the rootdn to also authenticate, as the rootdn is usually implemented as a organizationalRole, which has no uid attribute. Once the anonymous bind is successful, the user's dn is found, and a bind is attempted with the supplied password. If the bind is successful, the user has successfully authenticated.

Methods in this class simply allow for human-readable implementations of information needed from the class, for example 'is this user authenticated?'.

The class that implements the methods needed for session authentication and

[8] Note that the code could allow uids to clash, but this code is not implemented in this example. The code would loop through all found dn's until it found one that was able to bind with the supplied password. A glitch in the design would create an issue if two people had the same uid and same password – an argument for making uid unique throughout the directory.

authorization is the ldapUser class, implemented in the file ldapUser.php.

```
<?php
class ldapUser {
//
// Copyright © 2003 Justin B. Alcorn
//  All Rights Reserved
/*
*   This script is  free software;
*   you can redistribute it and/or modify
*   it under the terms of the GNU General
*   Public License as published by
*   the Free Software Foundation;
*   either version 2 of the License, or
*  (at your option) any later version.
*
*  The GNU General Public License can be found at
*  http://www.gnu.org/copyleft/gpl.html.
*
*  This script is distributed in the hope
*  that it will be useful,
*  but WITHOUT ANY WARRANTY; without even
*  the implied warranty of
*  MERCHANTABILITY or FITNESS FOR A PARTICULAR
*  PURPOSE.  See the
*  GNU General Public License for more details.
*/
//
// Based on an idea from http://martin.f2o.org/php/login
//
   var $ldadb = null; // ldapDB pointer
   var $failed = false; // failed login attempt
var $date; // current date GMT
var $dn = null;
var $logged = 0;
var $attrs = array();

function ldapUser(&$ldapdb) {
   $this->ldapdb = $ldapdb;
   $this->date = $GLOBALS['date'];
// Does the user have an existing session?
   if ($_SESSION['logged'] & $_COOKIE['ldapUserTimeout']) {
      $this->_checkSession();
   }
   return $this;
}

// checkLogin
//     authenticates the user against the LDAP server
//
function checkLogin($username, $password) {
   $newdb = new ldapDB($this->ldapdb->host,
                        this->ldapdb->base,
                        $username,$password);
   if ( $newdb->isAuthUser() ) {
      error_log("We have a good logon",0);
      $this->ldapdb = $newdb;
      $this->dn = $newdb->binddn;
      $this->logged = 1;
      if ($res = ldap_search($newdb->conn,
                        $newdb->base,
                        "(uid=".$username."*)",
                  array("uid","cn","sn","givenName","title"))) {
         if ($entry = ldap_first_entry($newdb->conn,$res)) {
            $this->attrs =
                     ldap_get_attributes($newdb->conn,$entry);
         } else {
            $this->attrs = array();
         }
         $saveattrs = serialize($this->attrs);
         $_SESSION['attrs'] = $saveattrs;
         $_SESSION['dn'] = $this->dn;
         $_SESSION['logged'] = true;

         $this->_updateCookie();
         return true;
      } else {
         error_log("LDAP err".ldap_error($this->ldapdb->conn),0);
      }
   } else {
      error_log("Failed LDAP Bind for $username $password",0);
      $this->failed = true;
      $this->logout();
      return false;
   }
} // checkLogin

//
// inGroup - return true if $this->dn is a member of $group,
//  or a member of any group within $group.
//     Function is recursive.
//
function inGroup($group) {
   $dnparts = ldap_explode_dn($group,0);
// all is a pseudo-group.  All, including anonymous,
// are in this group.
//      Make sure cn=all,ou=group,o=<org> is in the dir.
   if ($dnparts[0] == "cn=all") {
      return true;
   }
//allusers is a pseudo-group that includes anyone logged in.
//    make sure cn=allusers,ou=group,o=<org> is in the dir
   if ($this->logged && $dnparts[0] == "cn=allusers") {
      return true;
   }
// users who are not logged in.
//   again, create the pseudo-group
   if ($dnparts[0] == "cn=anonymous") {
      if ($this->logged) {
         return false;
      } else {
         return true;
      }
   }
   if ($this->dn) {
      $srch = "objectclass=*";
      if ($res = @ldap_read($this->ldapdb->conn,$group,$srch))
         {
         if ($e = @ldap_get_entries($this->ldapdb->conn,$res))
         {
            $entry = $e[0];
            if (in_array('groupofnames',
                           $entry['objectclass'])) {
               $i = 0;
               while ($i < $entry['member']['count'] ) {
                  if ($this->dn == $entry['member'][$i]) {
                     return true;
                  } else {
                     if ($this->inGroup($entry['member'][$i])) {
                        return true;
                     }
                  }
                  $i++;
               }
            }
         }
      }
      if (ldap_errno($this->ldapdb->conn)) {
         error_log("inGroup:".$group.":".ldap_error($this->ldapdb->conn),0);
      }
   }
   return false;
}      // inGroup

//
// appPerm - return true if $this->dn a part of any
//  group listed in the members of the $app object
//
function appPerm($app) {
   if (!$app) {
      return false;
   }
   $srch = "objectclass=*";
   $sbase = "cn=".$app.",ou=Apps,".$this->ldapdb->base;
   if ($res = @ldap_read($this->ldapdb->conn,$sbase,$srch)) {
      if ($e = @ldap_first_entry($this->ldapdb->conn,$res)) {
         $entry = @ldap_get_attributes(
                  $this->ldapdb->conn,$e);
         $j = 0;
         while ($j < $entry['member']['count'] ) {
            if ($this->inGroup($entry['member'][$j])) {
               return $entry['labeledURI'][0];
            }
            $j++;
         }
      }
```

```
    }
    if (ldap_errno($this->ldapdb->conn)) {
       error_log("appPerm:".$app.":".ldap_error($this->ldapdb-
  >conn),0);
    }
    return false;
 } // appPerm

 //
 // listApps - return a list of apps that the user is
 //    allowed to access.  Useful for building a dynamic
 //    menu
 //
 function listApps() {
    $srch = "(objectClass=labeledURIObject)";
    $sbase = "ou=Apps,".$this->ldapdb->base;
    if ($res = @ldap_search($this->ldapdb->conn,$sbase,$srch,
                                array("cn","labeledURI"))) {
       if ($entries = @ldap_get_entries(
                         $this->ldapdb->conn,$res)) {
          $i = 0;
          while ($i < $entries['count']) {
             if ($this->appPerm($entries[$i]['cn'][0])) {
                $apps[] = $entries[$i]['labeleduri'][0];
             }
             $i++;
          }
       }
    }
    if (ldap_errno($this->ldapdb->conn)) {
       error_log("listApps:".ldap_error($this->ldapdb-
  >conn),0);
    }
    return $apps;
 } // listApps

 //
 // logout - destroy the session, set all variables to defaults
 //
 function logout() {
    $_SESSION['logged'] = false;
    $_SESSION['dn'] = null;
    $this->logged = 0;
    setcookie('ldapUserTimeout',0,time()-86400);
 } // logout

 //
 // _updateCookie - utility that sets the timeout cookie
 //
 function _updateCookie() {
    setcookie('ldapUserTimeout',1,time()+(60*30),'/');
 } // _updateCookie

 //
 // _checkSession - make sure the session has not been hijacked
 //
 function _checkSession() {
      // This is where a session can be 2xchecked.
      // For these purposes, if the session exists, we're logged
in
      if (TRUE) {
         $this->dn = $_SESSION['dn'];
         $this->logged = 1;
         $this->attrs = unserialize($_SESSION['attrs']);
         $this->_updateCookie();
      } else {
         $this->logout();
      }
   } // _checkSession
   }

   ?>
```

The constructor for ldapUser takes as a parameter an existing ldapDB object, and checks to see if the user has an existing session. If there is no session, the object is created and returned. Otherwise, the session is checked. In this example implementation, we assume that if the session exists it is valid. Note, however, that PHP session hijacking is a known possibility, and any production implementation should have code in the _checkSession utility function that would double check the validity of the existing session. The cookie used is a idle timeout, and is updated with each call to the object. Absolute timeouts can also be implemented by setting a time-limited cookie at login and checking for it's existence.

New logins are handled by the checkLogin method. We create a new ldapDB object using the authentication parameters. If the ldapDB object returns as a authenticated user, the user is now logged on and we store the relevant information about the user in the ldapUser object. We also save certain attributes useful to applications in the PHP session variables. If the website wants to use a 'remember me' option for logins, a third parameter should be implemented here and some information should be saved in a persistent store, allowing a permanent cookie to authenticate a user. The constructor method can then check for this cookie. This should not be implemented for any site where the authetication and authorization is used for sensitive information.

The other methods implemented in ldapUser are used for the authorization function of the class. The inGroup() method is used to determine if the current user is a member of the named LDAP group. The function is recursive, so if the user is a member of any subgroups, the method will return true. Method appPerm() is used to determine if the current user has authorization to use the named application, as listed in the ou=app LDAP subtree. The app's 'member' attribute contains the distinguished names of each group that has authorization to use that application. appPerm() uses the inGroup() method to determine if the user should be allowed to use the application.

The method listApps() uses appPerm() to check each listed application and return a list of applications that the current user is authorized to use. This method is useful to build a dynamic menu of usable applications.

Using the class objects

To simplify securing any PHP applications on a web server, a PHP include file can be used. As an example, the following file, 'secure.php', can be placed in the PHP include path:

```
<?php
//
//  based on an idea from http://martin.f2o.org/php/login
//
require_once "ldapDB.php";
require_once "ldapUser.php";

$LOGIN_PAGE = "/index.php";

// Start the session
session_start();
session_register();

$date = gmdate("'Y-m-d'");
$ldapdb = new ldapDB("localhost","o=jalcorn.net");
$user = new ldapUser($ldapdb);

if (! $user->logged ) {
        session_defaults();
        if (! $noredirect == 1) {
                header ("Location: ".$LOGIN_PAGE);
      exit;
        }
}
if ($needauth && (! $appname || ! $user->appPerm($appname))) {
       header("Location: ".$LOGIN_PAGE);
       exit;
}

function session_defaults() {
   $_SESSION['logged'] = false;
   $_SESSION['dn'] = '';
   $_SESSION['cookie'] = 0;
}

?>
```

The code here starts the PHP session, creates the anonymous bind to the ldap server, and instantiates the ldapUser object. If the user has an authenticated PHP session, the ldapUser object will have the user's information, with an anonymous bind to the LDAP server.

If the user is not logged on, any session parameters related to the authentication are set to the defaults, and the user is redirected to the page to login, unless an

$noredirect parameter is set. If the user is required to be authorized to use the application, as signaled by the $needauth variable, and is not authorized, the user is redirected again. The login page need not be secured, but if can be with the following code snippet:

```
1. <?php
2. $noredirect = 1;
3. $appname = "loginform";
4. require_once "secure.php";
5.
6. if (! $user->logged  ) {
7.             // display login form for non-authenticated users
8.          exit;
9. }
10.    if (! $user->appPerm($appname)) {
11.         //code For authenticated, but not authorized users
12.    } else {
13.         // Code for authorized users
14.    }
15.    .. .  remainder of code, for all authenticated users
```

Line 2 tells the application not to redirect if the user is not logged on. The secure.php then setus up the user information, but does not redirect when the user is not logged on. Then the application checks to see if the user is a logged on user, and if not displays the correct form and exits.   Line 10 then checks to see if the current user is authorized. If not, some code is executed that could display an error message, redirect the user, reset the session, or provide some other information.

For regular, secured applications, the $noredirect parameter is not set. Users who attempt to execute the application without authenticating will be redirected to the login form by the 'secure.php' include file. For example:

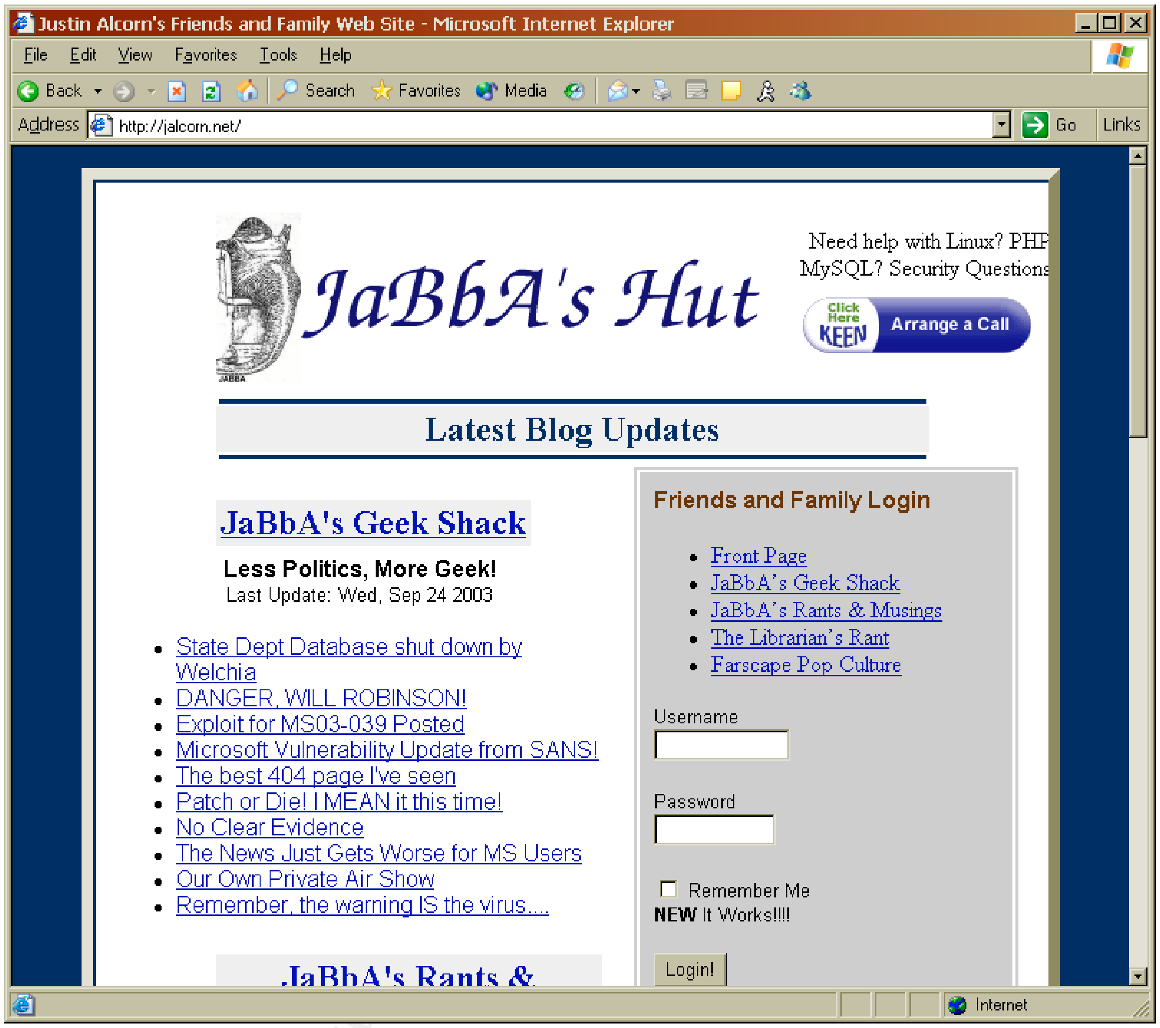
```
<?php
   $appname = "myapp";
   $needauth = 1;
   require_once "secure.php";

   … application code
```

Note that the application need not do anything about authentication or authorization. It is all handled by the 'secure.php' include file and the objects.

The Results - After

This code has been implemented as a prototype and example at http://jalcorn.net. Authentication and authorization is used to provide access to Family photos, personal blogs, calendars, a contacts directory, and administrative tools. Upon accessing the webpage as an unauthenticated user, a login screen is presented:


Justin Alcorn's Friends and Family Web Site - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://jalcorn.net/
Go Links
Click Here KEEN Arrange a Call
Internet

JaBbA's Hut

Need help with Linux? PHP MySQL? Security Questions

Latest Blog Updates

JaBbA's Geek Shack

Less Politics, More Geek!
Last Update: Wed, Sep 24 2003

- State Dept Database shut down by Welchia
- DANGER, WILL ROBINSON!
- Exploit for MS03-039 Posted
- Microsoft Vulnerability Update from SANS!
- The best 404 page I've seen
- Patch or Die! I MEAN it this time!
- No Clear Evidence
- The News Just Gets Worse for MS Users
- Our Own Private Air Show
- Remember, the warning IS the virus....

JaBbA's Rants &

Friends and Family Login

- Front Page
- JaBbA's Geek Shack
- JaBbA's Rants & Musings
- The Librarian's Rant
- Farscape Pop Culture

Username

Password

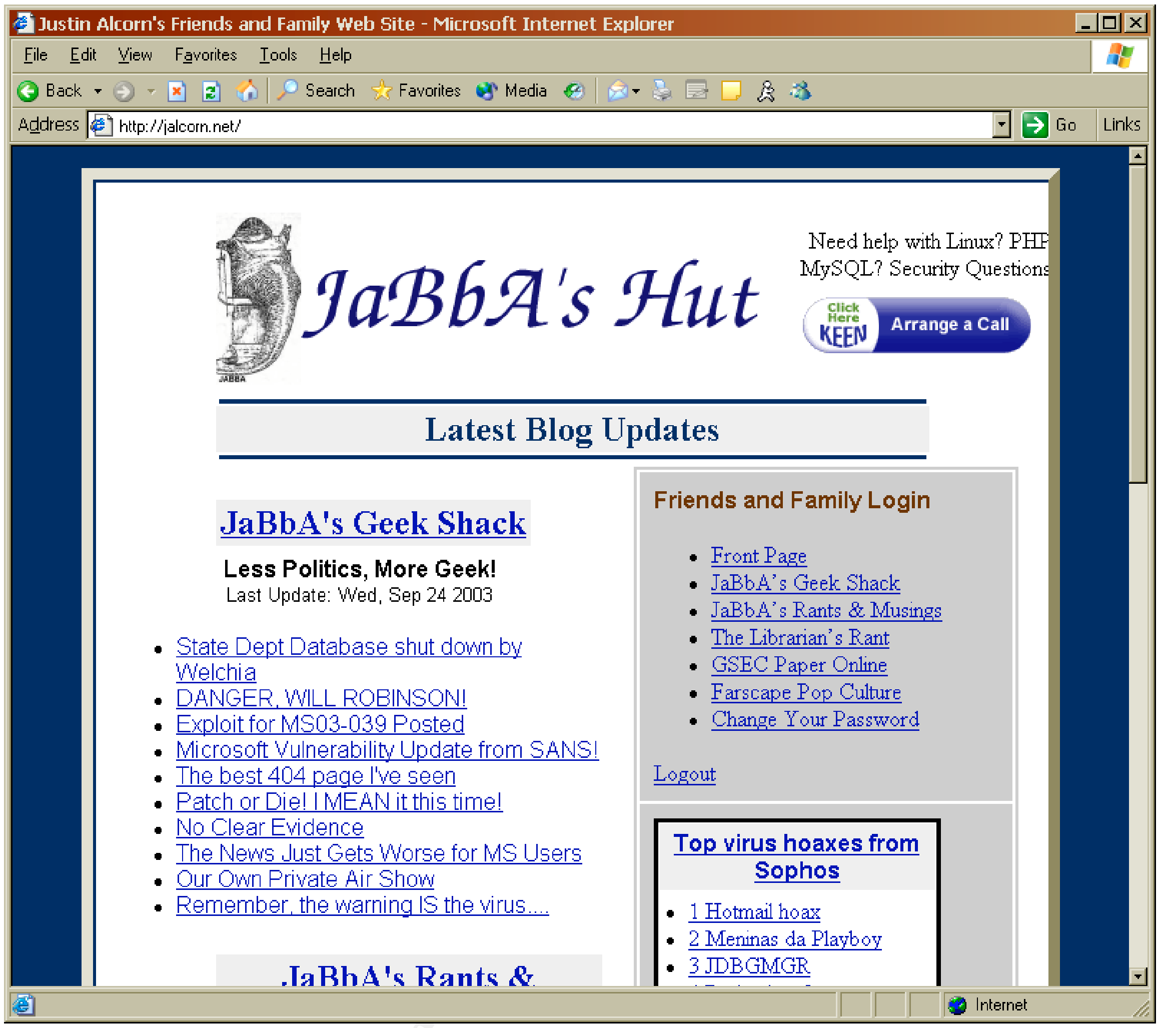
Remember Me
NEW It Works!!!!

Login!

This paper was made available to authenticated users by saving it as html, renaming it 'index.php', and adding the following code to the top:

```
<?php
$appname='gsecpaper';
$needauth=1;
require_once "secure.php";
?>
```

The app 'gsecpaper' has been included in the directory with authorization allowed for 'allusers'. A user has been set up with username 'example' password 'example'. (Note that the Change Your Password application has a some hardcoded code that disables it's functionality for the example user). Note that the menu of applications, built with the listApp() method, changes on login.


Justin Alcorn's Friends and Family Web Site - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://jalcorn.net/
Go Links
JaBbA's Hut
Need help with Linux? PHP MySQL? Security Questions
Click Here KEEN Arrange a Call
Internet

Latest Blog Updates

JaBbA's Geek Shack

Less Politics, More Geek!
Last Update: Wed, Sep 24 2003

- State Dept Database shut down by Welchia
- DANGER, WILL ROBINSON!
- Exploit for MS03-039 Posted
- Microsoft Vulnerability Update from SANS!
- The best 404 page I've seen
- Patch or Die! I MEAN it this time!
- No Clear Evidence
- The News Just Gets Worse for MS Users
- Our Own Private Air Show
- Remember, the warning IS the virus....

JaBbA's Rants &

Friends and Family Login

- Front Page
- JaBbA's Geek Shack
- JaBbA's Rants & Musings
- The Librarian's Rant
- GSEC Paper Online
- Farscape Pop Culture
- Change Your Password

Logout

Top virus hoaxes from Sophos

- 1 Hotmail hoax
- 2 Meninas da Playboy
- 3 JDBGMGR

ToDos

The classes presented here are usable in their present form, but should have more checks to counteract the possibility of session hijacking and other application attacks. Absolute timeouts, 'remember me' functionality, and IP address checking are all ideas that can be incorporated into these classes. Some of these changes have been made in the live code running on the prototype systems.

Impact

After prototyping the code, it was implemented on development systems in a coporate environment with the same tools. The applications that were written for the MySQL database were changed to implement the LDAUser class. Static webpages were secured by changing them to PHP code, as the GSEC paper was, and other applications were rewritten or front-ended with PHP code. All of these ad-hoc applications, which had previously been running on various machines scattered around the department, were now brought together, along with public security weblogs and virus warning applications, into a IT Security intranet. Communications with the end user population has significantly improved with this tool, and other IT personnel, including the Help Desk, can use some of the applications to find information and build new resources.

During the process of testing different commercial LDAP servers, we imported the authentication information and pointed our IT Security intranet's authentication information to the test servers. In most cases, The LDAP authentication worked with the commercial products as easily as with the OpenLDAP implementation. When we ran into problems with the implementation, it pointed out some places where the directory's implementation of the LDAP protocol may have been somewhat non-standard, and this information was documented and used as part of the decision-making process.

The use of open source tools has allowed us to solve problems in creative ways, and it's adherence to what will soon be corporate standards allows us to expand our possible toolset to solve any future problems to include open source tools. Management, which had orignally never considered open source tools for applications building, is now willing to look at open source alternatives when making buying decisions. This project, originally undertaken as a quick proof of concept, has opened the doors to new ways of solving problems.

Conclusion

Using an LDAP server and these objects provides a way for PHP application developers to write secured applications that can used centralized, LDAP provisioned usernames and passwords. In this way, corporations can begin to leverage the flexibility and availability of PHP source code without losing centralized provisioning and strength of a corporate directory server. These PHP examples can easily extended to Perl, Java, C++ and other languages, and corporations can then build applications in the languages best suited to the task.

References

Sheresh, Beth and Sheresh, Doug. Understanding Directory Services 2000 New Riders Publishing

Wilcox, Mark Implementing LDAP 1999 Wrox Press

Carter, Gerald LDAP System Administration 2003 O'Reilly & Associates

JasonSmith, Michael P. "LDAP Authentication and Session HOWTO" http://ldots.org/ldap/

M. Smith “Definition of an X.500 Attribute Type and an Object Class to Hold Uniform Resource Identifiers (URIs)” http://www.ietf.org/rfc/rfc2079.txt

M. Wahl, T. Howes, and Kille, S. “Lightweight Directory Access Protocol (v3)” http://www.ietf.org/rfc/rfc2251.txt

M. Wahl “A Summary of the X.500(96) User Schema for use with LDAPv3” http://www.ietf.org/rfc/rfc2256.txt

J. Franks, P. Hallam-Baker, et. al. “HTTP Authentication: Basic and Digest Access Authentication” http://www.ietf.org/rfc/rfc2617.txt

Martin Tsachev “Creating a Secure PHP Login Script “ http://martin.f2o.org/php/login

John Coggeshall “Session Authentication” http://www.zend.com/zend/spotlight/sessionauth7may.php

Mark Burnett “Auditing Web Site Authentication, Part Two” http://www.securityfocus.com/infocus/1691

O’Reilly & Associates “Web Database Applications with PHP & MySQL” http://www.wdvl.com/Authoring/Languages/PHP/WebDBApps/

Netcraft, Ltd, “Netcraft” http://news.netcraft.com