World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:5, No:6, 2011

W-CAS: A Central Users Authentication and Authorization System for Enterprise Wide Web Applications

Sharil Tumin, Sylvia Encheva

Abstract—Centrally controlled authentication and authorization services can provide enterprise with an increase in security, more flexible access control solutions and an increased users' trust. By using redirections, users of all Web-based applications within an organization are authenticated at a single well known and secure Web site and using secure communication protocol. Users are first authenticated at the central server using their domain wide credentials before being redirected to a particular Web-based application. The central authentication server will then provide others with pertinence authorization related particulars and credentials of the authenticated user to the specific application. The trust between the clients and the server hosts is established by secure session keys exchange. Case-studies are provided to demonstrate the usefulness and flexibility of the proposed solution.

Keywords—Authentication, Authorization, Security, Protected Web-based Applications.

I. Introduction

A person becomes an authenticated user of a system by providing correct credentials during the process known as sign-on process. These credentials are normally in the form of a pair of user-identification and password, shared between the authenticating system and the user. At the process of sign-on the person will be identified (i.e. providing a valid user-identification) and authenticated (i.e. providing the correct password in relation to the given user-identification). A user is signing-on into a system in order to make use of an application or accessing a resource. Authentication only perform users validation. Access controls and the related permissions to applications and resources are done by the process of authorization. Access and action permissions are normally defined by domain wide groups memberships and roles.

Authentication preceded authorization, in another word, to be authorized one must be authenticated first. In simple application, general permissions are given when a user is authenticated, without the need for further authorization processes. Normally, permissions are given at different points in an application in relation to what groups and roles an authenticated user is a member of and has. Permissions are granted or denied dynamically in relation to user's work flow within a Web-based application and are defined within the application. Groups and roles are providing authorization data of a user. Mapping these information in to permissions is done by the applications.

In this article we describe a system for a central authentication and authorization services for enterprise wide Web-based applications called *W-CAS*, that provides centralized services for security clearance of users, groups and roles information within an organization.

The main purposes of the *W-CAS* system are:

- To provide centrally managed sign-on and access control mechanism.
- To provide flexible and customisable access control front-end solutions for all Web-based applications.
- To increase security by protecting users' credentials and permissions on a single authoritative server.
- To reduce complexity by introducing simple process patterns for users' authentication and authorization.
- To promote and increase users' trust to the enterprise Web-based systems.

Nowadays, Web-based applications and services are common within any enterprises as a part of their ICT (information and communication technology) infrastructure. Web-based applications and services are used to provide computing and information services for; 1) the employees internally, 2) business partners externally, and 3) general public globally. Many of these services are deployed as protected Web-based applications secured by users authentication and authorization. These secure Web sites pose management and engineering problems on how to securely manage users' credentials and permissions and to ensure secure mechanism for sign-on and on-demand permits granting procedures. Users located anywhere geographically can connect to these secure Web sites at any time.

The *W-CAS* is a combination of secure Web-based applications and service, deployed on a secure server with a well publicized URL (Uniform Resource Locator) and generally known throughout the entire enterprise providing users with a consistence sign-on Web form. Clients (users' Web browsers) are using secure HTTP (Hypertext Transfer Protocol) to communicate with the server. The clients are HTTP redirect to the *W-CAS* from a protected Web-based application by using Web redirection mechanism. By creating session keys for each authenticated user, the *W-CAS* can provide authorization data to a protected Web-based application on-demand at a latter time in a user's work flow.

The *W-CAS* can also provide different types of SSO (Single-Sign-On) services for the entire enterprise. SSO can be em-

S. Tumin is with IT Department, University of Bergen, PO Box 7800, 5020 Bergen, Norway, e-mail: edpst@it.uib.no

S. Encheva is with Faculty of Technology, Business and Maritime Sciences, Stord/Haugesund University College, Bjørnsonsg. 45, 5528 Haugesund, Norway, e-mail: sbe@hsh.no

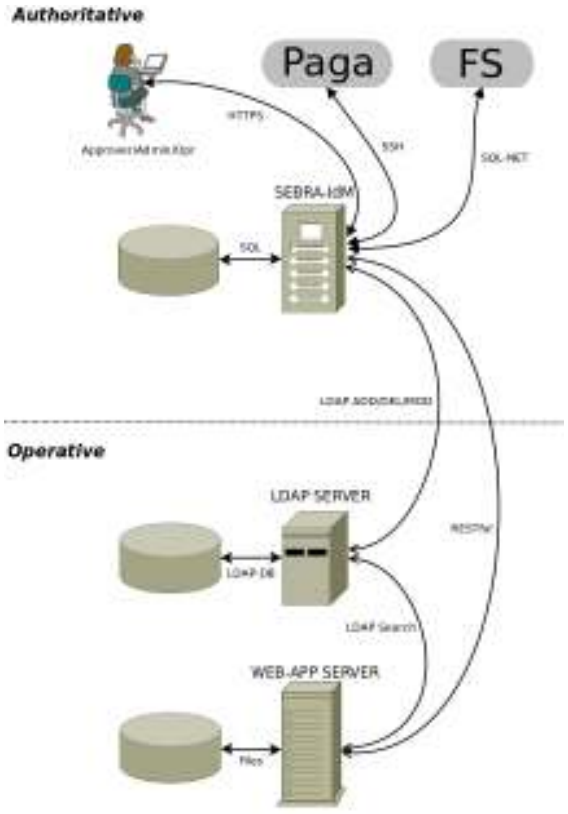

Fig. 1. SEBRA: IdM at University of Bergen

ployed; 1) to implement OT (one time) sign-on (SSO-OT) and using session keys or tickets to access multiple protected independent software systems without the need of further authentication, and 2) to implement a single CM (credential mapping) sign-on (SSO-CM) mechanism on different systems using different locally defined credentials and authentication.

We include in this article two case-studies of practical use of *W-CAS* in an ICT environment of a medium size university, one case describing SSO-OT and the other describing SSO-CM. The examples in the case-studies are included here to demonstrate the usefulness and flexibility of *W-CAS* in providing simple solutions to different authentication and authorization needs within an organization.

II. Background

A central IdM (Identity Management) system is a necessary condition within an organization in order to provide a system for a central authentication and authorization services.

The ideas of CAS (Central Authentication Service) are not new. There are several different CAS systems already successfully implemented at different sites around the world. These different implementations are providing solutions to different but similar problems related to Web-based authenticator and SSO.

A. Identity Management

An IdM is a centrally managed database concerned with identifying individuals (i.e. system users) within different IT (information technology) systems for the entire enterprise. An IdM deals with:-

- *persons information and user accounts*: Accounts are associated to persons, preferably one account to a person. In order for a person to access and use system resources, a user account must be assigned, normally by associating a user-identifier with that person. The user-identifier will be used to identify the person to the system. A user-identifier is a short alphanumeric string, e.g. 'ntu0675' and 'edpkm'.
- *accounts' credentials*: While user-identifier is for user identification, user's password is for user authentication. This (*user-id*, *user-pwd*) pair is normally used as credential. To be authenticated, a person provides her *user-id* to identifier herself and her *user-pwd* to prove that she is who she said she is. Users' passwords are not stored in clear text in the IdM databse, the passwords are stored as one-way cryptographic hashes. e.g. unix-crypt, MD5, SHA1, and SHA255.
- *accounts' roles and permissions*: User authentication is just the first half of an access control mechanism. The second and very important half is user authorization. Most authorization mechanisms are based on users' groups memberships. A particular group is directly mapped to a particular role. All users having that particular role will be defined as member of the group. Permissions are granted by the applications to users based on users' groups memberships.
- *accounts' security policies*: Implement password compositions and password aging policies. Send warning to users to change their passwords at appropriate time saying that accounts with aged password will be disabled. Keeping track of accounts periods and timely disabling of overdue accounts. Manage users authorization policies with the help of groups' membership. Since authorization policies are closely related to the IT resources and applications, the management of groups and roles are delegated to the local resources and applications managers. A distributed management model implemented as a Web-based application has proved to be a success.

Conceptually, it is easier to look at an IdM as a system with two main components; 1) *Authoritative*, and 2) *Operative*, Figure 1. The main function of the *Authoritative* component is to gather authoritative data from external data source; 1) *Paga* (employee's data), 2) *FS* (students' data), and 3) manually approve data by approvers and administrators for person not registered in *Paga* or *FS*, for example guest researchers. The main function of the *Operative* component is to provide operational authentication and authorization data to the IT systems within the organization by providing RESTful (Representational State Transfer) services from the IdM server or LDAP (Lightweight Directory Access Protocol) search services.

For authorization processes depending on complex data containing nested structure needed for authorization, RESTful services are preferred. Normally a simple LDAP search is sufficient for a Web-based application to determine user's

authorization at any stage of application's work-flow. The *W-CAS* is implemented to provide authorization data as a part of a user validation process.

B. Similar Systems

The *W-CAS* is very similar to some other implementations of CAS, Jasig-CAS, OpenID, Shibboleth, Pubcookie, JOSSO, SAML, CoSign, WebAuth, CookieAuth, and OpenAM [5], of which two will be briefly discussed in the following.

1) Jasig-CAS: Jasig-CAS [2] is primarily an authentication system written in Java and originally created by Yale University to provide a trusted way for an application to authenticate a user. Yale-CAS became a Jasig (Java Architectures Special Interest Group) project in December 2004. Jasig-CAS is widely used by educational institutions, especially many big universities.

Yale-CAS works using two client redirects (1,2) and one HTTP GET (3):

1) `https://s.y.z/cas/login?service=http://a.y.z/app`
2) `http://a.y.z/app?ticket=WXYZ`
3) `https://s.y.z/cas/serviceValidate?service=http://a.y.z/app&ticket=WXYZ`

If a valid credential is provided at stage (1), the CAS server would redirect the user's browser to URL described in (2). Now if the CAS server at stage (3) received valid *service* and *ticket* from the client then the CAS server would respond with the authenticated user identification from the sign-on process at (1).

2) OpenID: OpenID [4] was created by an open source community in the summer of 2005. Its prime concern is to provide an open standard on how users can be de-centrally authenticated. Users may choose any OpenID provider to consolidate their digital identities. Web-based applications have no need to provide their own authentication services. Several large organizations either issue or accept OpenIDs. These including among others, Google, Facebook, Yahoo!, Microsoft, AOL, MySpace, Universal Music Group, France Telecom, and Telecom Italia.

To have the benefit OpenID services, a user must first register to a OpenID provider. Here the user chooses her credentials, commonly a pair of user-identifier and password. The user also provide personal information which may or may not be true. OpenID does not check the validity of information given by users. OpenID assumes users truthfulness.

OpenID authentication works as follows:

1) An unauthenticated user provides her OpenID (e.g. `bob2pent.myopenid.com`) to the Web-bases application (e.g. `see.gettapp.com`) that supports OpenID authentication.
2) The application redirects the user to OpenID provider `myopenid.com`.
3) The provider checks the user's credential (i.e. user-identifier and password).
4) When validated, the provider will present the user with an OpenID verification, stating the application `see.gettapp.com` and what personal data the user allows to be shared with the application.
5) The provider `myopenid.com` redirects the authenticated user back to the application `see.gettapp.com`.

Anyone can freely choose any (not already used) OpenID credentials and any OpenID providers. A user can freely choose any OpenID provider where she is registered for authentication at any Web-base application site that supports OpenID. There is no central authority that dictates these choices. A person can register with many different personals to one or many different OpenID providers. There is no central validation authority.

C. A Brief Look at Web Technologies

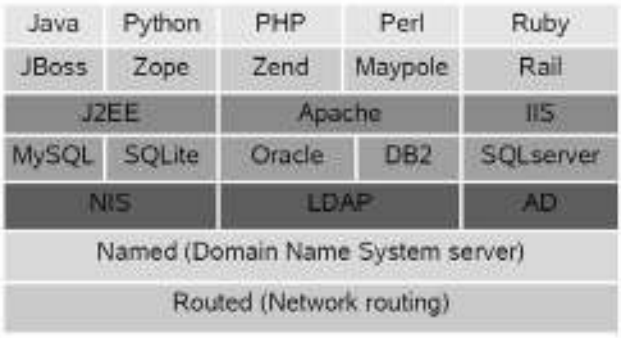

Fig. 2. Software Tools for Web Developments

The best way to present the Web Technologies used in deploying a Web-based system of applications is to summarize their relationships and dependencies in a technology stack diagram. Each row in Figure 2 shows different types of software technologies with examples of the different choices of tools. These technologies are arranged from bottom to top in these order; 1) Internet routing, 2) Internet naming services, 3) Directory services, 4) Database servers, 5) Web servers, 6) Web framework, and 7) Programming languages.

Users' credentials and groups memberships can be stored at and served from directory servers or database servers. The used of LDAP for directory services is more common in a Unix or Linux environment than in a Windows environment where the use of AD (Active Directory) is more prevalent. AD is a Microsoft customized LDAP with specialized schemas for Windows OS (Operating System). In an older Unix environment, NIS (Network Information Service) developed by Sun Microsystems Inc. (bought by Oracle Corp. 2010) can still be in use, albeit no longer as common as before.

There are diverse possibilities and combinations for Web developers to choose from in terms of technologies. Recently, many Web frameworks were being developed based on MVC (Model-View-Controller) concept, to help Web developers to design, develop, and deploy Web-based applications. Web frameworks provide developers and users alike with; 1) rapid

development cycles, 2) increase in operational security, and 3) sustainable deployment.

The successful deployment of Web-based systems depends very much on three technological innovations of our modern time; 1) the Internet, 2) the RDBMS (Relational Database Management Systems), and of course 3) the World Wide Web.

The Internet provides an open technological framework in which a computer can communicate with other computers over large distances across organizational and national boundaries spanning the whole globe. A database can provide a persistence data storage with relational structures for easy information storage, search and retrieval. The Web became popular due to the simplicity in its concept, design and implementation. The Web is modeled on client-server network architecture with stateless request-response mode of communication. The Web is built to provide easy and accessible tools for network programming.

These three technological innovations changed the way computer applications are perceived; 1) from a single user, single machine to multiple users and multiple machines, 2) from a stand alone application to collaborative applications, and 3) from a close private system to an open shared systems.

III. W-CAS

Enterprises are providing Web-based services and applications internally to employee and externally to costumers. Much of the traditional PC-based applications running standalone on personal computers are now replaced by Web-based applications. There are many reasons why these are so and among them are; 1) operative systems being independent, 2) centrally managed databases, 3) simplification of sharing of data, documents and reports, and) reduction of operational costs.

Web-based applications that replaced office automation programs for example need to be protected by some means of authentication and authorization mechanisms. All such Web-based applications have sign-on pages and presentation of these very much depends on the page style of the applications. These Web-based applications are deployed from 1) commercial softwares, 2) open-source softwares, and 3) in-house developed softwares. The authentication actions behind these sign-on pages are LDAP authentication mechanisms as shown in the *Operative* side of Figure 1.

There are many reasons why a centrally managed IdM is a good strategic security policy for both an organization and users. One of the reasons is that each user needs only to remember one credential. The same user-identification and corresponding password can be used for user's validation on all protected resources and applications deployed and managed by the organization. Due to the enterprise wide implication to security when a credential is lost or stolen, the issue pertinence to users' credentials protection is becoming more important.

One way to reduce the risk of credentials being stolen by phishing is to provide enterprise users with one and only one secure sign-on site for all enterprise Web-based application authentication procedures. The users will be informed and educated about the sign-on URL (Uniform Resource Locator),

Fig. 3. W-CAS: Sign-on

the page layout, the host and the host's HTTPS (Hypertext Transfer Protocol Secure) TLS (Transport Layer Security) certificate details. As an enterprise-wide security policies; 1) all users are informed not to provide sign-on credentials to other URLs than the official central authentication server, and when in doubt the users can examine the TLS certificate details, 3) all Web-based applications must use CAS for authentication and authorization by providing clients with the correct HTML (HyperText Markup Language) redirection.

A. Implementation Model

The authentication process model can be simply explained using a series of figures as shown in Figure 3. The basic

mechanism is similar to Yale-CAS and OpenID, the only differences are in the implementation details and focus areas.

While Yale-CAS and OpenID were designed to be as generic as possible, *W-CAS* is designed to solve specific problems. While Yale-CAS and OpenID are for use in an open community, *W-CAS* is for use in a closed organization. *W-CAS* uses similar pattern in the communication protocol, however the data communicated are specifically problems oriented and depend on the need of the client's Web-based application.

W-CAS is designed to satisfy both authentication and authorization needs for Web-based applications within an organization. User rights to a resource, service or application are permissions to perform certain actions in relation to certain conditions determined by users' roles at particular point in time.

$$Rights^{user}_{\{resources\}} = Permission_{\{actions\}} || Roles_{\{conditions\}}$$

Initially, an authenticated user has the general role of valid member of the organization, while more specific roles will depend on groups memberships. A valid member of the organization is an implicit role conditioned to any permits granting process of a resource or a group resources.

Given that $Right := \varrho$, $Permission := \varphi$, and $Roles := \rho$; an authenticated user *bob* having rights on resources α, and β with $read$ and $update$ permissions given that he is a member of A and B groups, can be symbolically represented as:

$$\varrho^{bob}_{\{\alpha,\beta\}} = \varphi_{\{read,update\}} || \rho_{\{A \cap B\}}$$

In this scenario, *W-CAS* provides the Web-based application with:

1) $authentication$: validation process that validate, *bob* is *bob*.
2) $authorization$: validation process or provide data for checking that $bob \in A \cap bob \in B$.

Note that $\rho_{\{A \cap B\}}$ is equivalent to $\rho_A \rho_B$, and $\rho_{\{A \cup B\}}$ is equivalent to $\rho_A | \rho_B$. Where the role condition is true when both are true for $\rho_A \rho_B$ and the role condition is true when either one or the other is true for $\rho_A | \rho_B$. This is to emphasis that the Web-based application can check role conditions sequentially, one condition at a time.

B. Implementation Details

The *W-CAS* is implemented using simple combinations of; 1) session Web cookies, 2) HTML redirections, and 3) RESTful query. Both the *W-CAS* server and the clients need to follow a prescribes protocol conceptually described in Figure 3. The *W-CAS* server is implemented in *Python* programming language with the support of *Apache* Web server module *mod_python*. In the examples, the client side is implemented in *PHP* programming language.

When referring to the numbered stages as shown in Figure 3, please find some detail explanations corresponding to the following numbered items:

1) *Is user login?* There are several ways to check whither a user is authenticated (i.e. login) and one common way is to check for session cookie returned by the Web client. However, the first page of Web-based application normally assumes that a user is not yet authenticated.
2) *No! Redirect to W-CAS* – The Web application will calculate a *hello* session cookie. Together, the cookies and the redirection directive to *W-CAS* server using 'Location are sent to the Web client (i.e. users Web browser) in the HTML headers. The browser will save the *hello* session cookie and be redirected to *W-CAS*. See Listing 1.
3) *Correct credential?* The user is now interacted with the *W-CAS* which asks the user to provide correct credentials (i.e. user-identification and password) for user validation. Security policy on how many retries are allowed can be enforced at this stage. Anyway the user needs to provide correct credentials in order to proceed.
4) *Yes! Redirect to Web App, W-CAS session check* – The user is now authenticated. The *W-CAS* calculates a *W-CAS-Session* for this sign-on and redirects the user's browser back to the Web application server at a specific URL declared in the configuration at *W-CAS* together with the calculates session. The *W-CAS-Session* is an encrypted (BlowFish) data containing all the necessary information in order to establish trust communication protocol with the Web application server. See Listing 2.
5) *Correct session?* The main purpose of this Web page at the application server is to receive the *W-CAS-Session* by the redirection and to send it back the *W-CAS* using RESTful query to a specific URL. The user will not even notice this action. The user will be redirected to the protected application if the session verifies to be correct one. See Listing 3.
6) *Yes! User is now authenticated* – The send back *W-CAS-Session* will then be decrypted and all its contents are unpacked. At the Web application initial session *hello*, the user-identification and the user's role are returned. The Web application will then make a session to store the authenticated user and redirect the user's browser to the protected application. The user can now continue using the application. See Listing 4.

The initial *hello* session sent from the application's *login* page to the *W-CAS* is to establish trust between the application and the authentication server. In the final stage of the protocol, the *W-CAS* returns the same *hello* with the authenticated user-identification and user role. Since communication between the application server and the *W-CAS* are done over HTTPS then there is minimum risk for "man in middle attack". Other programs can not sent a fake user authentication confirmation (i.e. *hello:uid:grp*) to the application server since the *hello* session is not known to them.

In this implementation the *W-CAS-Session* is an encrypted data containing all the information to securely protect the authenticated user-identifier by matching this data with IP-address of the application server, IP-address of Web browser, timestamp and the value of *hello*. The *W-CAS-Session* is

Fig. 4. MachForm: Sign-on

Fig. 5. MachForm: Authentication Mechanism

produced by login *W-CAS:login* and consumed by *W-CAS:auth* triggered by RESTful query from application server. The query is parameterized by the application name and *W-CAS-Session*. The key to decrypt *W-CAS-Session* is found in the configuration indexed by the supplied application name.

IV. Case-Studies

In this section we describe case studies of two actual applications employing *W-CAS* mechanism for authentication and authorization. In the first case, *W-CAS* is used as the basis for SSO-OT, while in the second case *W-CAS* is used to support SSO-CM. The first case is a commercial software with source code with possibility for customizable work. The second case is a commercial software with close code and there is no possibility for customization.

A. Case 1: Machform

Machform from Appnitro [3] is a Web-based application which provides user with Web-based UI (user interface) for designing Web form. Non technical users can design and publish nice looking and usable Web forms quickly and easily without the need to know HTML, CSS (Cascading Style Sheets) and JavaScript.

In the original, the product supports single user only. A lot of work has been done to convert a single user application into multi-users application that also supports ACL (Access Control List) mechanism for sharing Web forms among users with different permissions and roles; 1) admin - all permissions, 2) designer - update permission (i.e. read and write), and 3) friend - read only permission. A few Web pages in the work flow loop is shown in Figure 4.

The application is written in PHP with PostgreSQL database back-end. In order to support multi-users environment simple modifications were made on; 1) user authentication 2) data model that will support Web forms sharing and ACL. There are no local users defined in the application. All users with membership to a specific group defined in the IdM are allowed to access the application. The *W-CAS* is used to authenticate users and to verify group membership. The details of the sign-on is shown in Figure 5.

B. Case 2: Voicemail

The *VoiceMail* is a part of telephone services at a VIP2000 telephone system. The *VoiceMail* provides a Web-based application similar to telephone-based dialing for voice mail where one picks up the phone and dials a service number and then provides four digits PIN code. Similarly, in the *VoiceMail* Web-based application, a user provides her telephone number and the same four digits PIN to a Web form to get access to the telephone services provided by VIP2000 system.

The *VoiceMail* is a closed commercial system, which is aggregated by a collection of a compiled binary code, JavaScript and a MySQL database back-end. The only way to modify the interaction with the sign-on URL is by using HTML GET method instead of the HTML POST method via the provided Web form. To use *W-CAS* for authentication, the user-identification and password credentials need to be mapped to corresponding telephone number and PIN. Person telephone numbers are stored in the IdM but not the PIN codes, therefore in the *W-CAS* login form for *VoiceMail*, a user needs to provide; 1) user-identification, 2) password, and 3) PIN as shown in Figure 6. When the user verification is successful the *W-CAS* will redirect the user to the *VoiceMail* application together with the authenticated user's telephone number and the given PIN.

The *VoiceMail* application is deployed as a closed, stand-alone system without the possibility of supporting HTTPS. Communication between the browsers and the system is on an insecure channel where credentials (i.e. telephone number and PIN) are communicated in clear text. To increase the operational security, the *VoiceMail* application is placed behind a reverse proxy server (nginx) coupled with PHP-FPM (PHP FastCGI Process Manager) as shown in Figure 7.

V. Conclusion

This article demonstrates a practical deployment of CAS with two different applications. In an open source application, some simple customization needs to be done supporting the *W-CAS* protocol. In a closed code application, in order to use *W-CAS* authentication mechanism, users are diverted from the application sign-on page to the *W-CAS* sign-on page. After user verification the *W-CAS* must then do the sign-on on the

Fig. 6. VoiceMail: Sign-on

Fig. 7. VoiceMail: Authentication Mechanism

behalves of the user by sending either a HTML GET method or HTML POST method to the application's sign-on URL with all the needed credentials.

There are many well designed and well establish CAS systems already for an organization to choose from. However, these CAS systems may not satisfy the need of your organization for solving specific security problems imposed by local security policies and ever changing federal regulatory directives, for example mandatory installment of TFA (Two Factor Authentication) for certain Web application. With a basic *W-CAS* already deployed, a mechanism for a TFA can easily be implemented as an extension.

The best way for security engineers are to understand the basic concepts of CAS whereby a centrally authentication system can be designed and deployed to meet the organization operational security standards.

This article is an attempt to show how a CAS can be installed using these basic concepts and tools easily and effectively, with the possibilities for add-ons, thus supporting sustainable deployment.

APPENDIX A

Listing 1. Hello session and redirect to W-CAS

```
<?php
// login.php
// seed with microseconds
function make_seed() {
list($usec, $sec) = explode(' ', microtime());
return (float) $sec + ((float) $usec * 100000);
}
mt_srand(make_seed());
$hello = hash('ripemd160', mt_rand().mt_rand());
session_start();
$_SESSION['hello'] = $hello;
header("Location: https://cas.org/myapp/login
?app=myapp&hello=$hello");
?>
```

Listing 2. W-CAS session and redirect back to Application server

```
# login.py
if chk_auth(s):
    # create W-CAS session
    client = clientIP(r)
    appserver = apps[s['app']]['ip']
    appkey = apps[s['app']]['key']
    appurl = apps[s['app']]['url']
    ts = '%s' % int(time.time())
    hello = s['hello']
    key = appserver+client+ts
    key = key.replace('.', '0')
    b = Blowfish.new(key,
      Blowfish.MODE_CBC)
    sesid = s['user']
    data = b.encrypt(sesid+' '*\
      (8-len(sesid)%8))
    user_ses = binascii.b2a_hex(data)
    b = Blowfish.new(appkey,
      Blowfish.MODE_CBC)
    sesmsg = '%s:%s:%s:%s:%s' % \
        (user_ses, appserver,
        client, ts, hello)
    data = b.encrypt(sesmsg+' '*\
      (8-len(sesmsg)%8))
    ses = binascii.b2a_hex(data)
    uri = '%s?ses=%s' % (appurl, ses)
    # redirect back to application server
    util.redirect(r, uri)
    return
```

Listing 3. Welcome and Check W-CAS Session

```
<?php
// welcome.php
$ses = $_GET["ses"];
$base = 'https://cas.org/cas/myapp/auth';
$query = "app=myapp&ses=$ses";
$url = "$base?$query";
// get who is authenticated
$res = file_get_contents($url);
$prm = split(':', $res);
session_start();
$hello = $_SESSION['hello'];
// check hello cookies
if ($prm[0] != $hello) {
    echo 'Invalid session';
    exit;
// check valid user
if ($prm[1] == 'nobody') {
   echo 'Invalid user';
   exit;
// Authentication completed.
// Redirect to application
header("Location: https://myapp.org/app.php".
"?id=''$prm[1]."&grp=".$prm[2]);
exit;
?>
```

Listing 4. Check W-CAS Session

```
# auth.py
def index(r,s):
    nobody = 'NONE:nobody:nogroup'
    if not s.has_key('ses') or \
      not s.has_key('app'):
        r.write(nobody); return 'exit'
    else:
        appserver = apps[s['app']]['ip']
        appkey = apps[s['app']]['key']
        try:
            msg = binascii.a2b_hex(s['ses'])
        except:
            r.write(nobody); return 'exit'
        try:
            b = Blowfish.new(appkey,
                  Blowfish.MODE_CBC)
            ses = b.decrypt(msg)
        except:
            r.write(nobody); return 'exit'
        ses = ses.strip()
        try:
            s_user, s_appserver, \
            s_client, s_ts, s_hello = \
            ses.split(':')
        except:
            r.write(nobody); return 'exit'
        if appserver != s_appserver:
            # request from wrong app server
            r.write(nobody); return 'exit'
        try:
            msg = binascii.a2b_hex(s_user)
        except:
            r.write(nobody); return 'exit'
        client = clientIP(r)
        # s_ts can be use to enforce
        # period of validity check
        key = s_appserver+client+s_ts
        key = key.replace('.', '0')
        try:
            b = Blowfish.new(key,
                  Blowfish.MODE_CBC)
            uid = b.decrypt(msg)
        except:
            r.write(nobody); return 'exit'
        # all OK!
        grp = get_group(uid)
        rep_msg = '%s:%s:%s' % \
            (s_hello, uid, grp)
        r.write(rep_msg); return 'exit'
```

REFERENCES

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.
[2] Jasig, *Central Authentication Service Project*, http://www.jasig.org/cas, 2011 (last acc.).
[3] Machform, *PHP HTML Form Builder - Mailer Form Creator*, http://www.appnitro.com, 2011 (last acc.).
[4] OpenID Foundation, *Safe, faster, and easier way to log in to web sites*, http://openid.net/, 2011 (last acc.).
[5] Wikipedia, *Central Authentication Service*, http://en.wikipedia.org/wiki/Central_Authentication_Service, 2011 (last acc.).