Single Sign-On Research and Expansion Based On CAS

Fang Yinglan*, Jin Hao and Han Bing

Department of Computer, North China University of Technology, Beijing, China, 100144, P.R. China

Abstract: With the new application systems integrated with existing systems, many problems have arisen, such as system integration difficultly and audit function loss. Hence, Single Sign-on has become an important solution for popular enterprise business integration at present. According to the research and exploration of SSO system which is based on Yale-CAS authentication protocol, this paper introduced the integration of application systems using user-mapping with making the least amount of modifications. In addition, this paper has provided a complete security audit function for the single sign-on system, so a more comprehensive system security can be provided.

Keywords: CAS, security auditing, single sign-on, SSO, user-mapping.

1. INTRODUCTION

With the rapid development of computer network technology and information technology, more and more application systems are constantly being introduced, such as OA office systems, financial systems, resource management systems, personnel management systems and so on. However, with the passage of time and cross-business integration, many problems started to develop. Because each application has its own user management and authentication subsystem and maintains its own security policy, the user accessing different systems has to use different user credentials for independent authentication. In this process factors such as environment of the user, memory of the users and other such factors become a heavy burden on the users. On the other hand, the user authenticated information transmitted in the network causes a very serious security risk. To solve these problems, single sign-on concepts have come into being.

SSO is a convenient authorization mechanism for user to access multiple business systems. In a distributed environment, users only need to successfully login only once. Then they can freely switch between different systems. It allows users to save duplicate certification operations. This results in improved work efficiency, and reduces the burden on user's memory. In addition, single sign-on systems pay more attention on the entire certification process safety. The developer usually designs an encrypted function for the information transmitted in the network. This greatly enhances the security of the authentication process.

2. TRADITIONAL CAS AUTHENTICATION PROTOCOL

Currently, there are a lot of single sign-on implementation. Yale-CAS, Kerberos and Secure authentication models based on security assertion markup language (SAML) are common Single Sign-on models. Among these models, CAS is a secure authentication protocol developed by Yale University, it provides an enterprise solution for developers. It has a reasonable architecture and a rich interface support, also it supports cross-domain cookie sharing. This system makes an expansion of CAS authentication protocol to solve the complex business integration difficulties when achieving single sign-on system.

2.1. CAS Architecture

In structure, CAS single sign-on system is composed by the CAS Server and the CAS Client (Fig. **1**).

CAS Server needs to be deployed in an independent Java Web container. And the server becomes responsible for user's certifications. It runs on HTTPS protocol and consists of several Java Servlet. CAS Client can access the server through 3 URLs, named Login URL, Validation URL and Logout URL [1]. CAS Server has efficient configuration regarding management capabilities. It provides easy customizable authentication interface for developers, so that developers can freely extend authentication logic.

CAS Client is deployed with the application systems together, dealing with local protected resources requests. It filters requests and then redirects to CAS Server for user authentication, if the requests do not pass user certification.

CAS authentication protocol mainly focuses on designing a rational and efficient authentication module. Although user management features are closely related to a single sign-on system, in order to make the structure separately the CAS authentication protocol does not blend it together. It just provides an interface for developers to access the containers which store user data. User data can store in a relational database or a container based on lightweight directory access protocol (LDAP). When CAS Server needs user information, it can find it out in the configured storage.

2.2. CAS Authentication Process

The CAS system mainly uses tickets including TGC, ST and so on, to verify user's identity. A ticket-granting cookie

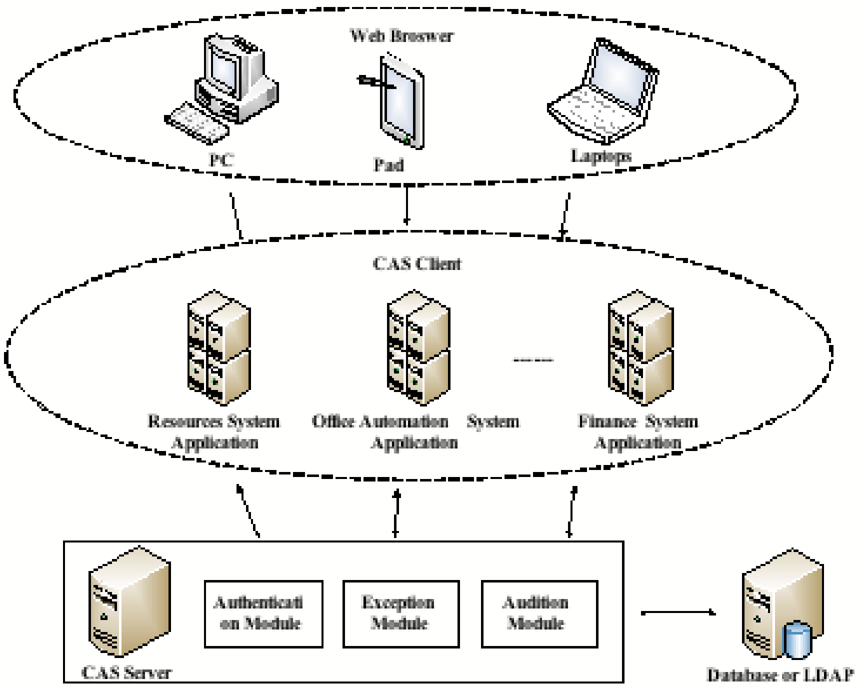


Fig. (1). CAS Single Sign-on Architecture Structure.

(TGC) is an HTTP cookie set by CAS upon the establishment of a single sign-on session. This cookie maintains login state for the client, and while it is valid, the client can present it to CAS in lieu of primary credentials. The value of ticket-granting cookies contains adequate amounts of secure random data so that a ticket-granting cookie is not guessable within a reasonable period of time. A service ticket is an opaque string that is used by the client as a credential to obtain access to a service. The service ticket is obtained from CAS upon a client's presentation of credentials and a service identifier [1]. CAS protocol certification process in shown in Fig. (**2**):

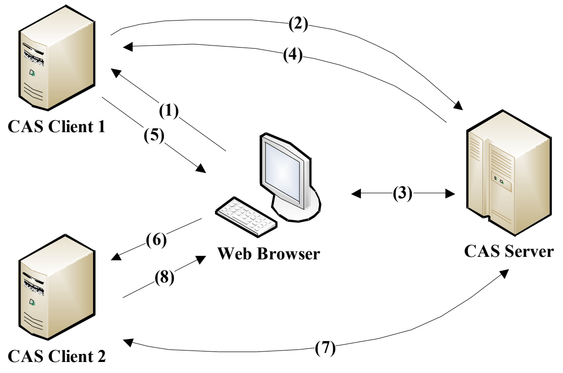


Fig. (2). CAS Protocol Certification Process.

1) When users first accesses the application's system protected resources, CAS Client1 checks the session and finds wether or not it has been created, and if the request does not contains a Service Ticket, all these factors indicate that the user is not authenticated.

2) CAS Client1 redirects to CAS Server to request a service ticket. Then CAS Server returns a service ticket and a service parameter which stores the application's system login URL.

3) When the CAS Server receives the CAS Client1's request, it will firstly try to find the ticket-granting cookie at user's browser. If there is not a ticket-granting cookie, CAS Server will redirect the user to the login page, then user will be asked to enter its credentials (username / password) to log.

4) After CAS Server issues a ticket-granting cookie, it find a service parameter in the request, after which it issues a service ticket and redirect the user to the application system which the service parameter points to.

5) When CAS Client1 receives the service ticket, CAS Client1 will validate it. If the service ticket is valid, CAS Client1 will create a session and provide the service.

6) When a user accesses the protected resources again, CAS Client2 finds that there is no session and no service ticket.

7) CAS Client2 redirects to CAS Server, then CAS Server finds that the user browser already has a ticket-granting cookie, so it issues a service ticket to CAS Client2 directly.

8) CAS Client2 validates the service ticket, creates a session, and provides the requested services of the user.

2.3. CAS Security Analysis

The concept of single sign-on itself avoids user's information being transmitted on the network frequently, and this greatly reduces the probability of losing information. CAS protocol also requires that all interactions between CAS Client and CAS Server should be conducted in SSL / HTTPS mode [2]. This transmission protocol is relatively safer and can maintain data integrity which prevents the user's information being intercepted. After user authenticates successfully, CAS will not pass any other user-related security information and interactions except for the username. As for the two important tickets: ticket-granting cookie and service ticket, CAS protocol also provides them a comprehensive consideration. First, the two ticket contain random data to ensure that the content is difficult to replicate. Secondly, the tickets will be encrypted before transmission. Finally, the two tickets have their reasonable survival period, even if the user does not logs off actively to destroy the two tickets, they will not be valid for a long term. CAS protocol also requires that service ticket be disposable, once the CAS Server validates a service ticket, it will be invalid immediately.

2.4. The Shortage of Traditional CAS Authentication

From the above analysis we know:

1) By design of CAS protocol, CAS Authentication Center provides unified authentication services. So the application system integrated into the CAS single sign-on system does not requires its own separate authentication module. Thus, CAS can take over the user authentication of new application directly, but as for the existing application systems, they usually have their own independent certification modules, and each application system's authentication is different. Therefore, if the developers wants to transform an existing application systems, it may involve changes to the system structure and processes. Such a transformation hides a huge risk, and it is very difficult too. In addition, some existing application system have no source code making the changes on the process modules impossible.

2) CAS Authentication Center needs to have its own user repository to store user credentials, but the original application systems also have their own user information database, the information's actual stored situation is very complicated, which makes it difficult to integrate user information together for a unified management.

3) CAS Authentication Protocol have no security audit function for the users' operations. Security audit monitors various events and behavior of the information systems, collects information, analyzes them and takes action for specific events and behavior. This includes the identification, recording, storing and analyzing of those activities related to information security [3]. Checking the results of the audit records can determine what happened to security-related activities and which users should be responsible for these activities. The single sign-on system which is an application of high-level security requirements should have the relevant security audit function.

3. RESEARCH AND IMPLEMENTATION OF SINGLE SIGN-ON BASED ON USER MAPPING

For new applications, developers only need to focus on its rights management and business implementation, as the authentication module can be directly handed over to the single sign-on system, thus avoiding duplication of the development tasks [4]. In addition, the user information database of new application systems can be unified, this also greatly reduces management costs. Considering that original application systems are difficult to make changes on and complex personnel information is difficult to be integrate with, we can make a map between CAS users and original application systems [5]. By mapping CAS Certification Center's user information and original application systems' user information, when the user certifies through CAS, the original application systems can get the user credentials to authenticate independently certified modules. Meanwhile, it is not necessary to integrate user information original application systems' user to the CAS user repository.

3.1. User Mapping Structure

Fig. (3) is a user mapping structure based on relational database. Among them, TAB_CAS_USER is CAS user information data table, and username is the user's unique iden-

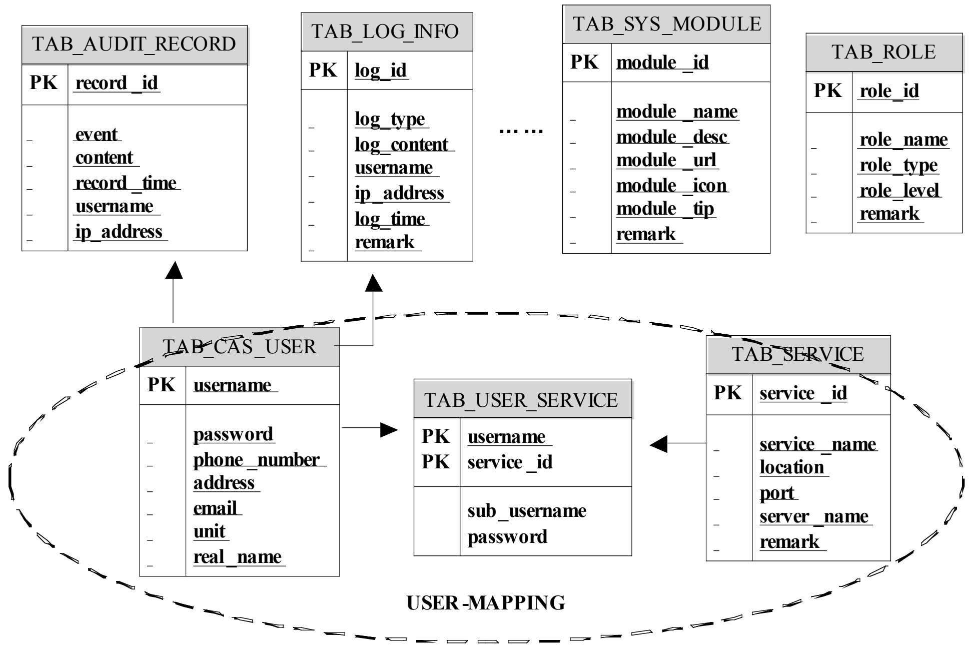


Fig. (3). User Mapping Structure.

tifier. TAB_SERVICE is a data table store protecting application systems information, and service_id is the system's unique identifier. Data sheet TAB_USER_ SERVICE makes users and service mapping, and the username and service_id are the primary key. Username and password is a user credential for the service which is identified by service_id.

Under such conditions, when the user has passed authentication, CAS Server can get all original user credentials through CAS username and return them to CAS Client. Next, developers can get user credentials according to the system identifier (service_id) to finish the auto-logging process [6]. Generally, auto-logging can be implemented by a plug-in or modification on the login page. In this way, developers no longer need to modify the source code to repeat the original authentication module.

3.2. The Design of Single Sign-on System Based on User Mapping

Mentioned in section 2.1, developers can customize their own authentication logic. They can configure custom authentication handler in deployerConfigContext.xml. The following is a part of the configuration for authentication handler:

```
<bean id="authenticationManager"
    class="org.jasig.cas.authentication.AuthenticationMa
nagerImpl">
    ......
    <property name="authenticationHandlers">
    ......
<bean
class="org.jasig.cas.adaptors.jdbc.CustomModeBata
baseAuthenticationHandler"></bean>
    ......
    </property>
</bean>
```

It should be noted that CAS authentication interface is used to provide users with a password encryption device (PasswordEncoder). Users can customize the password encryption. This system uses a salted md5 encrypted manner, taking several characters of username as "salt". This encryption method can ensure that the passwords stored in database are different from each other and can also effectively prevent attackers from getting another user's login credential according to the two same password string.

When a user logs in CAS Authentication Center, CAS Server redirects it back to the CAS Client to provide the user with services. Before redirecting, CAS Server requires user's credentials (username / password) of the application system feedback to the CAS Client, then application systems can finish their own authentication operation. Also in deployerConfigContext.xml file, developers can configure what data is taken which needs to be feedback to the CAS Client. Here are some content of the feedback configuration:

```
<bean id="attributeRepository"
class="org.jasig.services.persondir.support.jdbc.
MultiRowJdbcPersonAttributeDao">
    <constructor-arg index="0" ref="dataSource"/>
<constructor-arg index="1"
    value="select service_id, sub_username, password,
CONCAT(CAST(service_id as CHAR), CAST (username as
CHAR)) as 'pwdflag' from tab_user_service where {0}"/>
<property name="nameValueColumnMappings">
    <map>
    <entry key="serviceid" value="subusername" />
<entry key="pwdflag" value="password" />
</map>
</property>
<property name="queryAttributeMapping">
<map><entry key="username" value="user
name"/></map>
</property>
</bean>
```

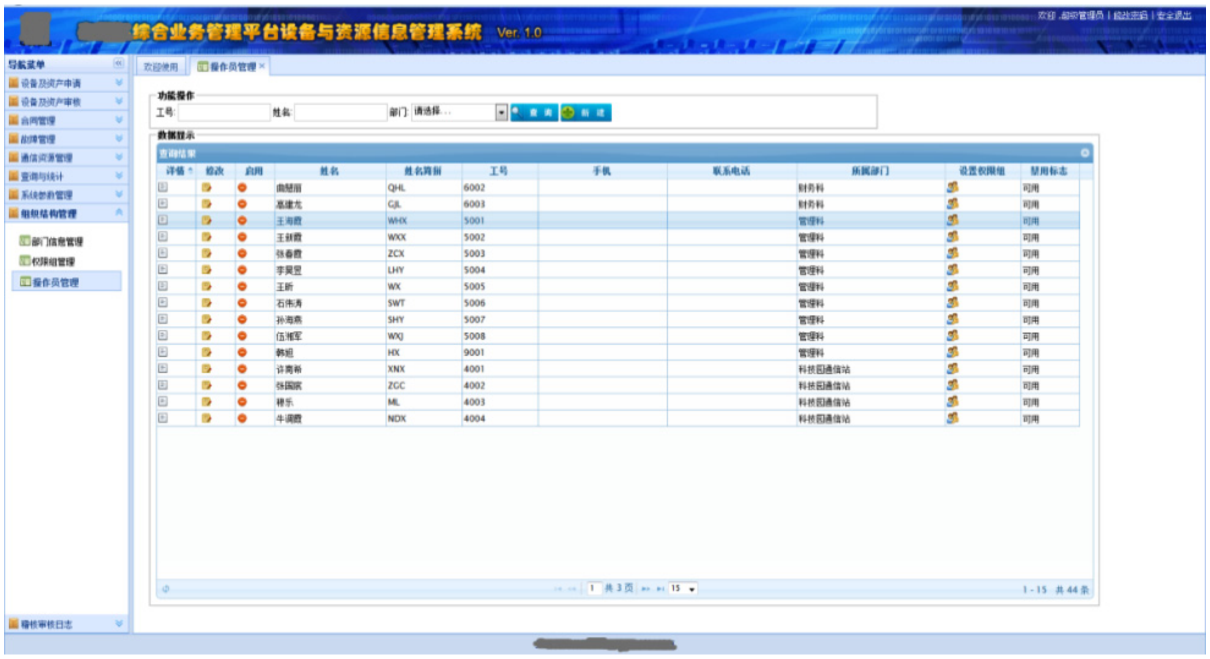
By CAS username, a user can obtain all of the original system credentials .So the sql statement which is the value of <constructor-arg index="1"> tag will query and gain several items. This data will eventually be in the form of key-value pairs (Map <K,V>) and returned to the client. Original application identifier (service_id) is the key for its username. The username for the original application system and its identifier consist of the key of its password.

On the CAS Client-side, by configuring CAS Assertion Thread Local Filter, users are able to obtain user credentials in Session which are returned from the CAS Server. After that, developers only need to modify the original login page of applications, and submit the user credentials directly to complete the auto-login operation [7].

Such a single sign-on design, developers can easily integrate the existing application services even if the integrated application services has no source code. Also, there are not any limitations for the introduction of new systems [8].

4. IMPLEMENTATION OF THE IMPROVED SYSTEM

Based on the above design, this paper used tomcat6.0 and CAS-Server-3.5.2 building an authentication server, and used MySQL5.6 storing for user-mapping of the data. Then we configured CAS Client on a call center system and a resource management system to integrate these two application systems on the client. We then configured the Login URL, Validate URL, Logout URL and CAS Assertion Thread Local Filter. So that when a user logs in, he can get org.jasig.cas.client.validation.Assertion object in session by using the "_const_cas_assertion_" parameter. The object carries the user mapping data, through the transformation of application system's login page we can submit the user credentials directly and achieve the function of original authentication module's auto-authentication. Fig. (**4**) shows one of the integrated application system's page of single sign-on system:


[redacted]综合业务管理平台设备与资源信息管理系统 Ver. 1.0

欢迎 ,超级管理员 | 修改密码 | 安全退出

工号: 姓名: 部门: 请选择...

详情	修改	启用	姓名	姓名简拼	工号	手机	联系电话	所属部门	设置权限组	禁用标志
			曲慧丽	QHL	6002			财务科		可用
			高建龙	GJL	6003			财务科		可用
			王海霞	WHX	5001			管理科		可用
			王新霞	WXX	5002			管理科		可用
			张春霞	ZCX	5003			管理科		可用
			李昊昱	LHY	5004			管理科		可用
			王昕	WX	5005			管理科		可用
			石伟涛	SWT	5006			管理科		可用
			孙海燕	SHY	5007			管理科		可用
			伍湘军	WXJ	5008			管理科		可用
			韩旭	HX	9001			管理科		可用
			许南希	XNX	4001			科技园通信站		可用
			张国辰	ZCC	4002			科技园通信站		可用
			穆乐	ML	4003			科技园通信站		可用
			牛迪霞	NDX	4004			科技园通信站		可用

1 共 3 页 15 1 - 15 共 44 条

[redacted]

Fig. (4). Resources Management System Interface.

4.1. Preparation

In order to extend CAS Server, we need to download the source code for a CAS Server on GitHub. As mentioned in section 2.1, the developers can customize their own authentication logic. According to different data storage container, they can configure custom authentication handler in deployerConfigContext.xml. For example, the system uses MySQL to store user data, we need to create an authentication processor in the cas-server-support-jdbc sub-project to achieve its own authentication.

It should be noted that, CAS authentication interface is used to provide users with a password encryption device (PasswordEncoder). Users can customize the password encryption [9]. This system uses a salted md5 encrypted manner, taking several characters with the username as "salt". This encryption method can ensure that the passwords stored in database are different from each other and can also effectively prevent attackers from getting another user's login credential according to two same password string [10].

4.2. The Deployment of CAS Server Authentication Center

CAS Server is a Web application, we need to deploy it in Tomcat. Normally, CAS Server uses HTTPS protocol to interact with CAS Client, so it is necessary to enable the security protocol by modifying Tomcat's server.xml configuration file. And before using HTTPS protocol, we need create a certificate and import it into JDK trusted repository.

1) RSA key generation command

keytool -genkey -alias cas -keyalg RSA -keystore c:/cas

2) Certificate Export command

keytool -export -file f:/resources/cas.crt -alias cas -keystore c:/cas

3) Certificate Import command

keytool -import -keystore %JAVA_HOME%\jre\lib\ security\cacerts -file c: /cas.crt -alias cas

Next, we need to add the following content to the server.xml file to enable HTTPS protocol:

<ConnectorSSLEnabled="true"clientAuth="false"keys_toreFile="c:\cas"keystorePass="changeit"maxThreads=" 150" port="8443"protocol ="org. apache.coyote.http11.apa_ che.coyote.http11.Http11 Protocol" scheme="https" secure ="true"sslProtocol= "TLS"truststoreFile ="%JAVA_HOME% \jre\lib\security\ cacerts "/>

As mentioned earlier, we have to customize a certification processor and a password encryption device, before the deployment of cas-server-webapp (ie CAS Server). They need to be configured. Developers can configure custom authentication handler in deployerConfigContext.xml. The following is a part of configuration content authentication handler:

```
<bean id="authenticationManager"
    class="org.jasig.cas.authentication.AuthenticationManagerImpl">
    ......
    <property name="authenticationHandlers">
    ......
    <bean
class="org.jasig.cas.adaptors.jdbc.CustomModeBatabase AuthenticationHandler" >
<property name="dataSource" ref="dataSource"></property>
<property name="passwordEncoder"ref="myMd5Pas_swordEnder"></property>
</bean>
```

```
    </property>
</bean>
<bean id="myMd5PasswordEnder"class="org.jasig.cas
.authentication.handler.myMD5PasswordEncoder">
 </bean>
```

Also in deployerConfigContext.xml file, developers can configure what data should be taken and from where it needs to provide feedback to the CAS Client. Here are some content of the feedback configuration:

```
<bean id="attributeRepository"
 class="org.jasig.services.persondir.support.jdbc.
 MultiRowJdbcPersonAttributeDao">
    <constructor-arg index="0" ref="dataSource"/>
<constructor-arg index="1"
value="select service_id, sub_username, password,
CONCAT(CAST(service_id as CHAR),
CAST(username as CHAR)) as 'pwdflag' from
tab_user_service where {0}"/>
<property name="nameValueColumnMappings">
    <map>
    <entry key="serviceid" value="subusername" />
<entry key="pwdflag" value="password" />
</map>
</property>
<property name="queryAttributeMapping">
<map><entry key="username"
value="username"/></map>
</property>
</bean>
```

By CAS username, a user can obtain all of the original system credentials. So, the sql statement which is the value of <constructor-arg index="1"> tag will become a query and gain several items. These data will eventually be in the form of key-value pairs (Map <K,V>) returned to the client.

When we finished these configurations, CAS Server can work properly according to our needs. We can access https://localhost:8443/cas to check whether the CAS Server works normally or not.

4.3. The Configuration of CAS Client

We integrate original application system with CAS Client through adding filters. In Java Web case, we add the following content in the web.xml file to access CAS Server's LoginURL, LogoutURL and ValidateURL these three service.

```
<filter>
<filter-name>CAS Single Sign Out Filter</filter-name>
<filter-class>
org.jasig.cas.client.session.SingleSignOutFilter
</filter-class>
</filter>
<filter-mapping>
<filter-name>CAS Single Sign Out Filter</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping>
<listener>
<listener-class>org.jasig.cas.client.session.Single Si-
gnOutHttpSessionListener
</listener-class>
</listener>
<filter>
<filter-name>CAS Authentication Filter</filter-name>
<filter-class>
org.jasig.cas.client.authentication.AuthenticationFilter
</filter-class>
<init-param>
<param-name>casServerLoginUrl</param-name> <pa-
ram-value>http://localhost/cas/login</param-value>
</init-param>
</filter>
<filter>
<filter-name>CAS Validation Filter</filter-name>
<filter-class>org.jasig.cas.client.validation.Cas20
ProxyReceivingTicketValidationFilter
</filter-class>
<init-param>
<param-name>casServerUrlPrefix</param-name>
<param-value>http://localhost/cas</param-value>
</init-param>
</filter>
<filter>
<filter-name>CAS HttpServletRequest Wrapper Filter
</filter-name>
<filter-class>org.jasig.cas.client.util.HttpServlet Re-
questWrapperFilter
</filter-class>
</filter>
<filter-mapping>
<filter-name>CAS Authentication Filter</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping>
<filter-mapping>
<filter-name>CAS Validation Filter</filter-name>
```

```
 <url-pattern>/*</url-pattern>
 </filter-mapping>
```

Furthermore, in order to get the application systems' user credentials passed from the CAS Server, we also need the following configuration:

```
<filter>
<filter-name>CAS Assertion Thread Local Filter
</filter-name>
<filter-class>
org.jasig.cas.client.util.AssertionThread LocalFilter
</filter-class>
</filter>
<filter-mapping>
<filter-name>CAS Assertion Thread Local Filter
</filter-name>
 <url-pattern>/*</url-pattern>
 </filter-mapping>
```

Through this configuration, we can get the map set which stores user credentials.

4.4. The Transformation of Original Application Systems' Login Pages

For the original application systems, we only need to transform their login pages. In JSP case, we can replace the original pages with a simple JSP page which only contains a <jsp:forward> tag. Then we configure the interlinkage, and get user credentials from session then submit them to the Servers.

5. COMPREHENSIVE ANALYSIS OF THE IMPROVED SYSTEM

5.1. Impact on the Original Application System

When we integrate application systems, we saved the authentication modules. Through modification of the login page, we can achieve the original application system auto-login, and make their login processes transparent to users. For the original architecture and business process we did not make any changes, so this design does not has any impact on the original application, this design also greatly reduces the cost and difficulty of businesses integrations.

5.2. Security Analysis of the System

The single sign-on extension mode of this system does not makes any changes for CAS protocol itself, and we do not repeal the authentication modules from the original application systems. We just transfer the original user credentials to CAS Client and put it into a session. Therefore, the only thing we should care about is the risk of leakage of user credentials. Although the original authentication modules still exist, the real user authentication is provided by CAS Authentication Center. So Even if the user credentials are lost during transmission, as long as they are different from CAS user credential, attackers still can't bypass the authentication. Therefore, this improved single sign-on solution still has the same security of CAS single sign-on system.

5.3. Unified Security Audit of the System

An excellent application security audit function can provide more comprehensive protection of information systems. Through recording the relevant security events and generating data, audit functions can help managers make effective analysis. Under the data analysis, managers can find out the reason why security breaching events happened. Most importantly, managers can take effective measures based on reasons of security breaching incidents to prevent the recurrence of such events. Different with other products or technologies, security auditing is ready before the incident, it makes a record when the event occurs, and provides an analysis of the data after the event has happened. It cares more about the whole process of events, rather than just providing protection from one angle or some stage.

For single sign-on system which requires high security, security audit function is particularly important. For the present improved single sign-on system, the operation related to safety such as user login, user password changing, user information maintaining and so on needs to be recorded and stored in the relational database.

In specific implementations, we need users' IP address. Usually, IP address can be obtained from Http-Requests, but the login operation is actually implemented in the authentication handler, and developers can't archive the object. However, after checking source code, we found that CAS stored user IP address in the object client information. So developers can obtain this object from com.github.inspektr.com_mon.web.ClientInfoHolder.

6. CONCLUSION

This paper describes the design and the expansion of CAS single sign-on system. We have achieved a unified authentication and integration for multiple applications. The integrated approach is based on user-mapping and can ensure that developers only need to modify pages for the original application system in order to integrate them with the single sign-on system. The source code is no longer necessary. Through the integration of multiple systems in the cluster, we have indicated that the solution does not only greatly reduces the difficulty of the original application systems integration but also the costs of integration. However, it does not make any integration limits for the new system. For the security audit function, it improves the security of the single sign-on system, and it also improves the fault tolerance of the system.

CONFLICT OF INTEREST

The authors confirm that this article content has no conflict of interest.

ACKNOWLEDGEMENTS

This work was partially supported by Natural Science Foundation of China (Grant No: 61070030, 61370051), It is also partly supported by Beijing Government and Education Committee (Grant No. PHR201107107).

REFERENCES

[1] D. Mazurek, "CAS Protocol", May 4, 2005[Mar 14, 2014], http://www.jasig.org/cas/prot-ocol

[2] Q. Zhang and G.B. Zhong, "Design and Implementation of CAS-SSO System Based on User-Mapping", *Research & Development*, vol. 4, pp. 7-10, 2009.

[3] Z.B. Tan, "*The design and implementation of single sign-on and behaviors audit system for remote desktop access*", Beijing University of Posts and Telecommunications, 2011.

[4] A. Armando and R. Carbone, "An authentication flaw in browser-based Single Sign-On protocols: Impact and remediation", *Computers & Security*, vol.33, pp.41-58, 2013.

[5] W. Z. Qiang, A. Konstantinov, D. Q. Zou and L. T. Yang, "A standards-based interoperable single sign-on framework in ARC Grid middleware", *Journal of Network and Computer Applications*, vol.35, pp.892-904, 2012.

[6] V. Radha and D. H. Reddy. "A Survey on Single Sign-On Techniques", *Proscenia Technology*, vol. 4, pp.134-139, 2012.

[7] Y.Q. Zhang, W. Chen, "Design and Implementation of CAS Single Sign-on System Based LDAP", *Software*, vol. 32, no.2, pp.14-17, 2011.

[8] R. Murri, P. Z. Kunszt, S. Maffioletti and V. Tschopp, "A single sign-on solution for grid web applications and portals", *Grid Computing*, vol.9, pp.441-453, 2011.

[9] W.J. Zhou, "Trial development of web-SSO model based on CAS", *Journal of ShangHai University of Engineering Science*, vol.23, no.2, pp.165-169, 2009.

[10] G.L. Wang, J.S. Yu and Q. Xie, "Security analysis of a single sign-on mechanism for distributed computer networks", *IEEE Transactions on Industrial Informatics*, vol.9, no.1, pp.294-302, 2013.

Received: September 22, 2014 Revised: November 30, 2014 Accepted: December 02, 2014