Approaches and challenges for a single sign-on enabled extranet using Jasig CAS

Florian Holzschuher, René Peinl

Institute of Information Systems – Hof University
Alfons-Goppel-Platz 1
95028 Hof
florian.holzschuher2@iisys.de
rene.peinl@iisys.de

Abstract: In this paper we describe our experiences with setting up a single sign-on enabled intranet with open source software and then making it accessible over the internet using a reverse proxy. During this process, we encounter several issues. We describe those, discuss possible solutions and present our final setup.

1 Introduction

Companies today often have a multitude of software systems running in their internal networks to support their business processes. For employees the ease of use increases, the more integrated these systems are. However, apart from Microsofts server systems that provide out-of-the-box single sign-on (SSO) in Windows domain environments, it is still not common to have even this basic integration, although Gartner called SSO as part of identity and access management (IAM) solutions a “must have for enterprises of all sizes and industries” [Wi+03] already in 2003. Especially in open source settings things seem complex since there are a large number of technological choices and no clear market leader, so that in many cases only authentication against a central LDAP directory is configured instead of SSO.

Our goal was to create an SSO-enabled extranet setup, making as few changes to the software used as possible. In our example, we connected Apache Rave, XWiki, Alfresco and Zarafa to Jasig CAS in order to provide single sign-on, based on accounts taken from a local LDAP directory service. This way, users only have one global account and only need to log in once per session, granting them access to all connected systems through their web browser. All systems do also share user profile information and report user activities to Apache Shindig in order to centrally display them in the Rave portal.

Jasig Central Authentication Service (CAS) was chosen as it is a relatively widely adopted open source authentication system, supporting multiple authentication protocols and methods. The rest of the paper is organized as follows. We first describe different SSO technologies and give an overview of some open source implementations. The we present our test setup. Afterwards we discuss general issues with the reverse proxy setup and specific issues with SSO, before we conclude with a discussion of results.

2 Single sign-on technologies

SSO can be applied in different scenarios that have different levels of complexity. The probably easiest case is given when all applications are running on the **Intranet** are using the **same runtime** environment like a Java application server and are **prepared for pluggable authentication** [Cl02] like using JAAS in Java, or SAP applications inside the Netweaver Application Server [Bo10]. In this case the container is managing authentication and authorization anyway, so it is quite easy to switch the container from the usual LDAP authentication to a central identity provider like CAS. Ideally, you don't have to make any changes on the application side. Pseudo SSO, using client-side technology to store passwords for server applications is not considered here [PM03]. SSO becomes more complicated if you are considering **multiple runtime environments** like one application running on Java, another on PHP and a third one on ASP.NET for example. You have to either find applications supporting authentication standards of the identity provider (see below) or an identity provider that supports all those (e.g., CAS). An additional level of complexity is added, if you are running your applications in an **extranet** setup [UB09], using a **reverse proxy** to relay and rewrite client requests that address a single host, to the multiple machines running the applications (see figure 1). The reverse proxy could be used to pre-authenticate the requests, so that only authenticated users are directed to the single applications [Ha+12]. Since CAS does not support all functionality with Apache Web server[1], we chose CAS filters inside Apache Tomcat running the applications.
A lot of research has been conducted on even more complex **federated scenarios** that enable users across organizations to access to access applications without additional login [CP07], [Ch09]. This requires one identity provider per organization and an established trust relationship, so that security tokens issued by one identity provider are trusted by all the others.

SSO can further be achieved using different authentication protocols. These are ideally transparently managed by the SSO system.
Kerberos is the predominant standard for SSO in Windows environments [Pr11]. In contrast to **NTLM**, which is the default authentication protocol for Windows, it is able to transfer credentials not only from client to server applications, but also down the road to further systems used by the service provider (e.g. the database). This feature is called delegated authentication. Kerberos can be used in Linux environments as well, although it is not trivial to setup the whole stack consisting of DNS (e.g. Bind), a certificate authority (e.g. OpenSSL), a directory service (e.g. OpenLDAP) and the core component key distribution center (KDC; e.g. Heimdal).
The Security Assertion Markup Language (**SAML**) is mainly used for authenticating against Web services. However, version two includes the Web Browser SSO profile

[1] https://wiki.jasig.org/display/CASC/Client+Feature+Matrix

(SAML SSO) designed to authenticate a user against Web applications [Hu+05]. It's an XML-based protocol designed to be included in transport means like SOAP over HTTP and already implemented by some large application providers like Google [Ar+11]. Besides service-oriented architectures (SOA) it is mainly discussed for cloud scenarios [Ce+10]. More recently, SAML is frequently accompanied by **XACML**, in order to provide attribute-based access control, which is a more general form of role-based access control [VDB07]. Emerging from public Web sites like social networks, **OpenID** was proposed as a means to use an identity from one identity provider for accessing other services [Io12]. However, in the internet OpenID currently suffers from relunctance of relying parties [Su+10] and lack of trust from end users [Su+11]. For authorization, OpenID is often accompanied by **OAuth 2.0** [SB12]. **OpenID connect** is a recent development in this area trying to better harmonize both parts [Bo12].

Finally, in the open source area, a multitude of SSO providers is available, each with tested compatibility to a number of different open source systems. A selection of well know open source SSO providers is briefly compared in table 1 and discussed below.

Table 1: comparison matrix for open source SSO solutions

	Jasig CAS	**JOSSO**	**WSO2 Id Server**	**Open AM**
Latest version	3.5.2 (22.02.13)	2.3.0 (31.08.12)	4.1.0 (11.02.13)	10.1.0 (20.02.13)
License	Jasigs own open source license	LGPL	APL v2	CDDL 1.0
Protocols	CAS, OAuth, OpenID, SAML, Kerberos	SAML, NTLM	OAuth, OpenID, XACML, SAML, … (18+),	OAuth, SAML, Kerberos
Authenti-cation backends	JAAS, LDAP, AD, Radius, JDBC, X.509, Negotiate (Kerberos)	JAAS, LDAP JDBC, two factor auth with WiKID, X.509	LDAP, AD, JDBC, Cassandra	LDAP, AD, two-factor auth with HOTP, Negotiate (Kerberos)
Runtimes	Tomcat or other Servlet 2.4 container	JBoss, Tomcat, Websphere, Geronimo, Jetty	WSO2 Carbon server	Tomcat, JBoss
Agents	Spring, MS IIS, JEE, Apache 2.2, PHP, PAM	Apache 2.2, PHP 4+, MS IIS, Liferay, Alfresco, phpBB, Coldfusion, Spring	None found	Apache 2.4, MS IIS, Sun Web Server, JBoss, Glassfish, Tomcat, Websphere, Web Logic

Shibboleth is the implementation of the Internet 2 consortium and specifically designed for federated scenarios [Ch09]. It uses SAML messages with digital signatures in order to improve trustworthiness (ibid.). It allows protecting the user’s identity by using random pseudonyms for every session. Since both federation and anonymity are not required in our scenario, we did not consider Shibboleth.
Jasig CAS (Central Authentication Service) is a SSO system using its own protocol (also named CAS). However, it also supports SAML and included support for SAML version 2.0 in CAS v 3.5.1 dating in October 2012 by updating to OpenSAML 2. It also includes support for OAuth 2.0 and can act both as an OAuth client and delegate authentication to other OAuth servers like Facebook or Google, as well as an own OAuth server to directly authenticate OAuth clients. The basic architecture of CAS is similar to the Kerberos model [Io12]. The CAS protocol also supports ticket proxying, which is similar to the Kerberos’ delegated authentication. Starting with CAS version 3, it does also support single logout [WY10]. We chose CAS due to its direct support of Liferay, Moodle and Mule.
Other open source SSO systems include **JOSSO** [AFG06], a completely Java based identity provider that also supports PHP and dotNET service provider and has a nice graphical tool to configure SSO scenarios, the **WSO2 Identity Server** [SFB10], which is especially interesting when using the family of WSO2 infrastructure products as well as the successor of the Sun OpenSSO framework **Forgerock OpenAM** [Th11]. We plan on testing some of these in future work. Especially OpenAM in conjunction with the Open Identity Gateway seems a promising alternative for our scenario.

3 Basic setup

Our setup consists of a single machine with an external IP running an Apache web server that acts as a reverse proxy, a single machine with Jasig CAS and several machines running our service providers, all of which are accessible through web interfaces. All machines are located inside a DMZ behind two firewalls, one towards the internet and one towards our internal network (see figure 1).

In contrast to common patterns [So03], we do not separate the proxy from the other machines by an additional firewall, but only use it as a gateway for terminating the SSL connection [Ma99]. The reverse proxy is using a signed certificate, only allowing HTTPS connections and redirecting any unencrypted calls. The Apache Tomcat instance running Jasig CAS is also SSL-enabled to allow for secure ticket validation, but is using a self-signed certificate. We are using AJP for connecting to Tomcat-based applications. However, that didn’t prove much better than a normal http connection (see section 4.2). We did also consider nginx as a replacement for Apache httpd since it is optimized for reverse proxy scenarios and provides an easy to use caching mechanism. Another alternative worth testing would be to use a specialized SSO gateway like the Forgerock Open Identity Gateway [BCN12]. It promises to enable SSO for those 30% of typical Web applications that do not work with the usual SSO filters or agents.

For all connected software systems running inside Apache Tomcat we used the authentication, validation, request wrapping and single sign-out filters provided by CAS.

These filters together redirect unauthenticated users to the SSO login page, validate incoming tickets and store the authenticated user in their respective sessions. The Apache web server used for the PHP-based Zarafa server is using a CAS authentication module (mod_auth_cas) which also redirects users, evaluates tickets and sets the logged in user for requests.

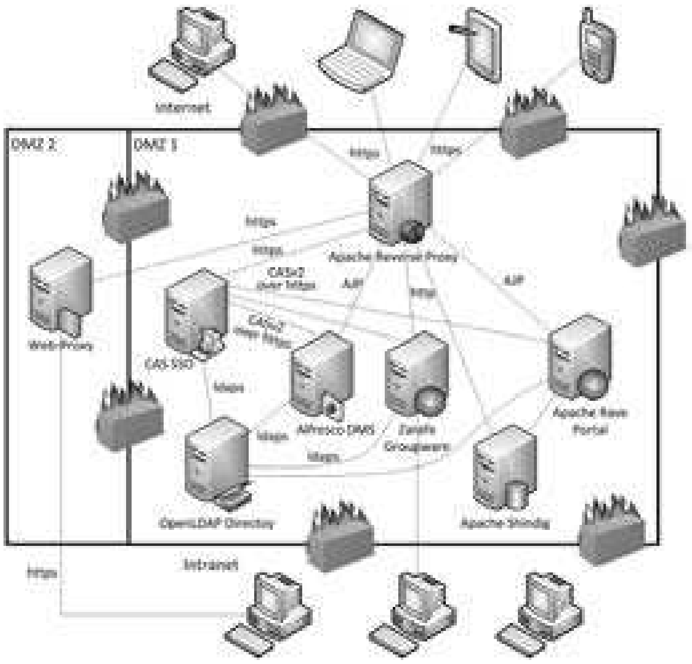

figure 1: system architecture of the extranet scenario

Yet, unless software is prepared for reading the provided session information, an authentication plugin is required, telling the service provider which user is currently logged in. Moreover, the first time a user logs in, a new local user account may have to be created, preferably using user data from the local LDAP server. A random local password should be set at that point to avoid empty local passwords, especially if local login cannot be disabled completely. Since we are using open source software exclusively, we were able to implement suitable plugins for almost all systems we wanted to include. However, in most cases configuration was enough and no programming was necessary. All systems are also connected to an LDAP server in order to retrieve additional user information like full name and email address from it.

We also conducted some tests with Android-based smartphones and tablets and found out that Chrome on Android behaves in the same way as its Desktop counterpart.

4 General challenges

Running our services through a reverse proxy and with single sign-on filters caused several problems, not all of which could be solved completely.

4.1 Platform problems

In general, the whole setup has more layers than a normal intranet setup, reducing performance noticeably. We addressed this by keeping rewriting to a minimum and only including paths in the SSO filtering that we were certain needed direct protection or an automatic redirect to a login page. When applicable we were able to achieve slightly better performance by using the binary Apache JServ Protocol (AJP) to connect application servers to the proxying web server, instead of a normal HTTP connection. It did also enhance performance to use nginx instead of Apache and enable its caching. However, the login process is not affected by this caching and nginx needs a plug-in for AJP support instead of supporting it natively.

Some applications had problems with being **accessed using HTTPS** in their external URL while the reverse proxy accessed them using HTTP. We were able to fix this by setting the appropriate parameters in all applications and placing redirects in our web server configuration. When using an HTTP reverse proxy, CAS' login page displays a **warning message about an unencrypted connection** that will not support SSO, but works anyway. We could eliminate this warning by using even an unencrypted AJP connection. Supplying all application and web servers with certificates and reconfiguring the reverse proxy to use SSL may also solve these protocol-related problems, but will result in a more complex setup with slightly lower performance.

In terms of usability, we found that unless directly connected via Spring Security, the applications' **logout buttons did not work with SSO**. They may terminate the application's own session, but with the SSO session still active, the user is immediately logged in again. This is especially confusing since CAS provides a dedicated single log-out (SLO) filter and the client feature matrix states that SLO should work out of the box for all clients, except Spring [Fr12]. It is also tough to correct this behavior programmatically. Although most systems provide an interface to create an authentication plugin, overriding the logout action is usually not available. In any case the question whether a user only wants to log out of a single application or terminate the whole SSO session is still open.

Beginning with Java 7, some **SSL warnings are treated as errors** and cannot be easily circumvented inside applications. We found that if the name in the certificate and the URL don't match, an **“unrecognized_name” error** is detected, which can be fixed by including all possible external server names as aliases in the web or application server's configuration. In this context, further problems can be caused by faulty DNS and domain name configuration, causing further **name mismatches on reverse lookups**.

4.2 Rewriting

Depending on which parameters are used for their generation, web pages delivered by proxied web services can contain **incorrect URLs** referencing other resources. This is caused by the Tomcat server detecting its own machine's external address, incorrectly specified external hosts and contexts differing from the local context inside Tomcat.

Since we were trying to run all services under sub paths of our external host to avoid URL collisions, we kept experiencing faulty redirects and incorrect links. One of our approaches was to use the Apache web server's rewriting functionality. It can be used to correct URLs in headers, links, cookies and references to and within other resources such as JavaScript and CSS.

As this approach did not produce consistently satisfactory results, we resorted to replicating the sub path structure on all web and Tomcat application servers. This way, we solved most problems concerning URLs and redirects and only had to manually correct some URLs in the applications' resources and configuration and move static resources to their new location if needed. The drawback of this method is having to create **static contexts in Tomcat**, with a hash tag in its name as a delimiter to denote the sub path. This was still necessary when using AJP instead of http as described below.

While using **AJP could largely solve the problem** of incorrect links and faulty redirects, we found that some applications' resources like CSS files and especially links within JavaScript code were still wrongly referenced. Therefore, additional rewrites or the sub path replication mentioned above were needed in this setup as well. We determined that this problem is based on the fact that applications, when started, use their local context information to generate resource links which are incompatible with the differing context used by the web server. We correctly configured the proxyName and proxyPort attributes in the AJP connector but still had those problems. The most notable benefit of AJP was, that CAS was no longer complaining about the unsecured connection between the proxy and the service providers when the “secure” option in AJP is set to true.

Another interesting configuration option in the AJP connector is called “tomcatAuthentication” and causes the **authentication to already be performed on the reverse proxy** instead of the connected tomcat application servers. This configuration looks similar to the Forgerock Open Identity Gateway solution. However, in our test, we could not perceive any notable differences compared to authentication using Tomcat, especially regarding performance. The configuration might become a bit easier though. Finally, you can configure encrypting the connection between Apache httpd and Tomcat by configuring a pre-shared key using AJP’s “requireSecret” option.

4.3 Service accessibility

Direct access to applications using their own **local administrative accounts**, did also prove to be a challenge. After the initial setup, those accounts are often the only way to properly configure applications and delegate permissions to other users. However, the filters used to protect services by redirecting unauthenticated users to the CAS login page are blocking access to the applications' own login mechanisms. We found no way to enable both SSO as a default method and still provide local access for admin users as a fallback. The only option looking promising in CAS is called **gateway mode**. This mode is attaching tickets to the request for already authenticated users and passing unauthenticated request through to the service provider. However, this mode requires larger changes to the service providers in order to start the SSO session.

One approach we took was **creating users with the same ID in the LDAP directory**, preserving for example the user's administration rights in the application. Alternatively, one could manually add or override these rights in the applications' account database for existing LDAP accounts, since the administration interface is unavailable. Of course, this approach is not necessarily suitable for production environments.

This could be circumvented by **storing per-application rights in the LDAP directory**, which some applications offer as an option. But this would require all necessary schemas to be incorporated into the directory's structure and additionally writing plug-ins for applications that don't already support this approach. Moreover, we reckon a dedicated authorization system might be a more elegant solution when dealing with a greater number of systems. Though, this will require more and possibly more complicated plugins, for which there is even less predefined support from applications.

Another way around this problem is a more **sophisticated authentication chain**, checking several authentication possibilities before redirecting the user and offering an opt-out functionality for the SSO mechanism. This way at least users knowing their full URL could still log in. For this to work, the SSO login page would need an additional opt-out button which redirects the user to the original page with an additional parameter. This parameter would then be detected by a modified SSO filter and disable or modify the redirect to allow a login.

Theoretically, one could also **disable the automatic redirects** altogether, making the user choose between a button to log in locally and one to log in using SSO, which triggers the redirect. This modification would be needed for each individual application and would make the fully automated sign-on procedure semi-automatic. To minizime the usability trade off made by this approach, at least within the comapny the SSO login page could be set as the browsers' starting page, offering the user to log in at the beginning of each session while still leaving the option to opt out.

Furthermore, services will also be used from within a company's network and concerning performance it would be desirable to **access them directly**, bypassing the reverse proxy. But we found that some applications need to have their external URL specified, which in our example would be pointing to the reverse proxy. Thus accessing applications directly can cause inconsistent web pages being generated, with resources being referenced internally as well as externally or possibly with an incorrect URL.

5 Specific problems

With Jasig CAS we encountered the problem that the **certificate** used for its Tomcat server needs to have the **subject alternate name** set correctly. While we could fix this for our self-signed certificates by generating them accordingly, CAS would not work with the existing certificate used by our reverse proxy server, since it lacks this parameter. To generate a suitable certificate using Java's keytool, Java version 7 is needed, so one needs to be careful when doing so as still many applications only work properly when using Java version 6.

When trying to connect our Apache Rave portal to its back-end, Apache Shindig, we realized that it was not easily possible to **authenticate against CAS** and maintain a SSO session, i.e. simulate a user, **from Java code**, to access the service protected by the CAS filters. The SSO setup we chose is designed to be used from a web browser and although the protocol is documented, we could not find or develop a connector that can establish a usable SSO session from java code. Similar problems will occur when communication between individual systems is required. Again, this could be resolved using a more sophisticated authentication chain, allowing other login methods to pass through without triggering redirects. We should also note that direct login from code using a username and a password is discouraged by the CAS developers, so our failure to maintain a session may be the desired behavior.

Otherwise **CAS' ticket proxying functionality** may provide a solution, also using a simulated client with a service account. This solution gives applications the possibility to request a proxy granting ticket for a logged-in user that it can use to request further tickets to be consumed and validated by other applications. This way, authenticated server-to-sever communication is possible without impersonating a user, assuming proxy tickets are accepted.

This is still not fully sufficient in our case as we also need to have server-to-server communication when there currently is no user logged in and background processes are firing events. Anyway, ticket proxying is more likely to work in our case since it only requires a single call to CAS from the user, causing a redirect to the calling service with the ticket needed to start proxying.

To enable this functionality we would need trusted, encrypted connections between the servers concerned, further HTTP service endpoints, capable of being validated by CAS and receiving proxy granting tickets and handling proxy tickets. Especially in case we wanted to use tickets from a real user session, we would also have to modify the applications' security systems, storing CAS session information so that it can be used from any part of the application requiring server-to-server communication.

Lastly, we tried to make some of our services searchable using **Apache Solr** with an unmodified ManifoldCF instance as a crawler. This also failed due to problems with the SSO session. As with our manual approach, we did manage to authenticate against CAS using built-in functionality but then failed to actually use the session for crawling. The crawler was being redirected back to CAS after each request, even though it specifically supports application server sessions.

6 Conclusion

Our experiences show, that adding only a little more complexity by introducing a reverse proxy leads to several issues with SSO in a real world scenario. Some of the issues described are application or SSO system specific, some others are only basic challenges like specific settings in the SSL certificate and a few are issues by design, like single sign-out.

We found that the difficulty of including an application into the whole setup depends strongly on the software design, which frameworks were used in what way and how well modifying an application's configuration, code and plugin capabilities are documented. For example using Spring Security offers generic interfaces for adding SSO support, which make integration rather easy in case you are familiar with Spring configuration. But we also found that in some cases an individual preparation for SSO plugins can be more suitable and easier to configure – possibly easier to handle in a more sophisticated authentication chain.

Our biggest problem with the reverse proxy setup was handling the context switch between application and web server. Many applications are able to detect requests through a reverse proxy or can be configured accordingly, dealing with protocol and external hostname changes. But we found that hardly any application will work normally when placed under a different path in the web server compared to the application server it is running on. This must be considered bad code design, since it is no problem to query the current context information like hostname and context path from the application server. However, using hard coded paths at least in some areas of the application (especially JavaScript) seems still the default, based on our tests. Maybe this is a negative side-effect from the current trend to port application code from the server to client-side JavaScript.

In general, we can conclude that a basic SSO-enabled extranet with CAS can be created with a reasonable amount of work, given well-prepared applications. Yet creating a well-rounded, high quality working environment will require extensive modifications to many applications' authentication systems and partially to the IT infrastructure around them.

References

[AFG06] Agostino Ardagna, C., Frati, F., & Gianini, G. (2006). Open Source in Web-Based Applications: A Case Study on Single Sign-On. International Journal of Information Technology and Web Engineering (IJITWE), 1(3), 81-94.

[Ar+11] Armando, A., Carbone, R., Compagna, L., Cuellar, J., Pellegrino, G., & Sorniotti, A. (2011). From Multiple Credentials to Browser-based Single Sign-On: Are We More

Secure? In: Future Challenges in Security and Privacy for Academia and Industry (pp. 68-79). Springer Berlin Heidelberg.

[BCN12] Bryan, P., Craig, M., Nelson, J. (2012): Guide to OpenIG 2.1.0. Forgerock. 17.05.2012 http://docs.forgerock.org/en/openig/2.1.0/gateway-guide/index/index.html

[Bo10] De Boer, M., Essenpreis, M., Garcia-Laule, S., Raepple, M. (2010):Single Sign-on mit SAP – Lösungen für die Praxis. Galileo Press, Bonn

[Bo12] Boyd, R.. Getting Started with OAuth 2.0. O'Reilly Media, Incorporated, 2012.

[CP07] Camenisch, J., & Pfitzmann, B. (2007). Federated identity management. In: Security, Privacy, and Trust in Modern Data Management (pp. 213-238). Springer Berlin Heidelberg.

[Ce+10] Celesti, A., Tusa, F., Villari, M., & Puliafito, A. (2010). How to enhance cloud architectures to enable cross-federation. In IEEE 3rd International Conference on Cloud Computing, pp. 337-345

[Ch09] Chadwick, D. W. (2009). Federated identity management. In: Foundations of Security Analysis and Design V (pp. 96-120). Springer Berlin Heidelberg.

[Cl02] De Clercq, J. (2002). Single sign-on architectures. In Infrastructure Security (pp. 40-58). Springer Berlin Heidelberg.

[Fr12] Fritschi, J. (2012): CAS Client Feature Matrix. Jasig. 04.04.2012. https://wiki.jasig.org/display/CASC/Client+Feature+Matrix

[Ha+12] Haron, G.R., Maniam, D., Sadasivam, V., Loon, W.H. (2012): Re-engineering of Web Reverse Proxy with Shibboleth Authentication. International Conference for Internet Technology and Secured Transactions, 10-12 Dec. 2012, London

[Hu+05] Hughes, J., Cantor, S., Hodges, J., Hirsch, F., Mishra, P., Philpott, R., Maler, E. (2005): Profiles for the OASIS Security Assertion Markup Language (SAML)V2.0. OASIS standard. http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf

[Io12] Ionita, M. G. (2012). Secure Single Sign-On using CAS and OpenID. Journal of Mobile, Embedded and Distributed Systems, 4(3), 159-167.

[Ma99] Maier, P. Q. (1999): Implementing and supporting extranets. Information systems security, 7(4), 52-59.

[PM03] Pashalidis, A., & Mitchell, C. J. (2003). A taxonomy of single sign-on systems. In Information Security and Privacy (pp. 249-264). Springer Berlin Heidelberg.

[Pr11] Pröhl, M. (2011): Kerberos – Single Sign-On in gemischten Linux/Windows Umgebungen. d.punkt Verlag, Heidelberg

[So03] Sommerlad, P. (2003): Reverse proxy patterns. In European Conference on Pattern Languages of Programming, EuroPLoP 2003.

[SFB10] Steuer Jr, K., Fernando, R., & Bertino, E. (2010). Privacy preserving identity attribute verification in windows cardspace. In Proceedings of the 6th ACM workshop on Digital identity management (pp. 13-16). ACM.

[SB12] Sun, S. T., Beznosov, K. (2012). The devil is in the (implementation) details: an empirical analysis of OAuth SSO systems. In: Proceedings of the 2012 ACM conference on Computer and communications security (pp. 378-390). ACM.

[Su+11] Sun, S. T., Pospisil, E., Muslukhov, I., Dindar, N., Hawkey, K., & Beznosov, K. (2011): What makes users refuse web single sign-on? an empirical investigation of OpenID. In Proceedings of the Seventh Symposium on Usable Privacy and Security (p. 4). ACM.

[Su+10] Sun, S. T., Boshmaf, Y., Hawkey, K., & Beznosov, K. (2010, September). A billion keys, but few locks: the crisis of web single sign-on. In Proceedings of the 2010 workshop on New security paradigms (pp. 61-72). ACM.

[Th11] Thangasamy, I. (2011) OpenAM. Packt Publishing, Olton

[UB09] Ullrich, M., & Rieger, F. (2009). Brancheninitiative Single Sign-On: Der sichere, einheitliche und einfache Zugang zu den Extranets der Versicherer wird Realität. In: Maklerveraltungsprogramme der Zukunft: Ein Ausblick auf zukünftige IT-Systeme zur Unterstützung von Versicherungs-und Finanzvertrieben, 179.

[VDB07] Vullings, E., Dalziel, J., & Buchhorn, M. (2007). Secure Federated Authentication and Authorisation to GRID Portal Applications using SAML and XACML. Journal of Research and Practice in Information Technology, 39(2), 101-114.

[WY10] Wang Y., Jia, Z. (2010): The Application Research of Single Sign Out Model Based On CAS. International Conference on Computer Design and Appliations (ICCDA 2010)

[Wi+03] Witty, R. J., Allan, A., Enck, J., & Wagner, R. (2003). Identity and access management defined. Research Study SPA-21-3430, Gartner.