FOCSE: An OWA-based Evaluation Framework for OS Adoption in Critical Environments

Claudio Agostino Ardagna, Ernesto Damiani, Fulvio Frati
University of Milan - via Bramante 65, Crema (CR), Italy
{ardagna,damiani,frati}@dti.unimi.it

Abstract. While the vast majority of European and US companies increasingly use open source software for non-key applications, a much smaller number of companies have deployed it in critical areas such as security and access control. This is partly due to residual difficulties in performing and documenting the selection process of open source solutions. In this paper we describe the FOCSE metrics framework, supporting a specific selection process for security-related open source code. FOCSE is based on a set of general purpose metrics suitable for evaluating open source frameworks in general; however, it includes some specific metrics expressing security solutions' capability of responding to continuous change in threats. We show FOCSE at work in two use cases about selecting two different types of security-related open source solutions, i.e. *Single Sign-On* and *Secure Shell* applications.

1 Introduction

In the last decade, open source operating systems and middleware platforms have been widely deployed [4]. In the security area, the adoption of open source solutions has been much slower, since most users do not completely trust the open source community and consider open source middleware a potential "backdoor" for attackers, potentially affecting overall system security. However, proprietary security solutions have their own drawbacks such as vendor lock-in, interoperability limitations, and lack of flexibility. Recent research suggests that the open source approach can overcome these limitations [3, 21]. According to some researchers, open source solutions may even in the end improve security, as they give greater visibility of software vulnerabilities [11], giving the possibility to fix them as soon as a threat is described. In our opinion, what is still missing to boost open source adoption in security is a selection framework allowing the users to evaluate the level of suitability of different open source security solutions. In itself, comparative evaluation of OSS is a time-honored subject, and several researchers [8, 12] have proposed complex methodologies dealing with the evaluation of open source

Please use the following format when citing this chapter:

Ardagna, C.A., Damiani, E., Frati, F., 2007, in IFIP International Federation for Information Processing, Volume 234, Open Source Development, Adoption and Innovation, eds. J. Feller, Fitzgerald, B., Scacchi, W., Sillitti, A., (Boston: Springer), pp. 3–16.

products from different perspectives, such as code quality, development flow and community composition and participation. General-purpose open source evaluation models, such as Bernard Golden's *Open Source Maturity Model* (OSMM) [14] do not address some specific requirements of security software selection. However, these models assess open source products based on their maturity, i.e. their present production-readiness, while evaluating security solutions also involves trying to predict how fast (and how well) a security component will keep abreast of new attacks and threats. A security-oriented software evaluation framework should provide potential adopters with a way to compare open source solutions identifying the one which *i)* best suits their current non-functional requirements and *ii)* is likely to evolve fast enough to counter emerging threats.

In this paper, we develop on our previous work [5] to obtain a specific technique for evaluating open source security software, including access control and authentication systems. Namely, we describe a *Framework for OS Critical Systems Evaluation* (FOCSE) based on a set of metrics aimed at supporting open source evaluation, through a formal process of adoption. FOCSE evaluates an open source project in its entirety, assessing the community composition, responsiveness to change, software quality, user support, and so forth.

The remainder of this paper is organized as follows. After a brief introduction of the basic concepts of comparative software evaluation (Section 2) we present our set of evaluation metrics (Section 3). Then, Section 4 presents the aggregator used to integrate different metrics in a single estimation, allowing for ranking analyzed solutions. Finally, Section 5 provides two use cases where the defined framework evaluates open source Single Sign-On (SSO) and Secure Shell (SSH) solutions. Section 6 gives our conclusions.

2 Basic Concepts

In this section, we provide a review of the technologies used in the context of FOCSE evaluation. In particular, we describe the `FLOSSmole` project [13] used to gather and store data about the open source solutions to be evaluated. An essential prerequisite of FOCSE is the availability of the raw data necessary to compute the metrics defined in Section 3. The availability of a database can greatly improve the reliability and the effectiveness of FOCSE. FLOSSmole (formerly OSSmole) [10,13] is a platform designed to gather, share and store data supporting comparative analysis of open source software. FLOSSmole is aimed at: *i)* collecting raw data about open source projects; *ii)* providing summary reports about open source projects; *iii)* integrate data from other research teams; *iv)* provide tools to gather the

data of interest. In the following analysis, we relied on FLOSSmole for collecting information about the projects subject to our analysis.[1]

3 FOCSE: a Framework for OS Critical Systems Evaluation

Generally speaking, few organizations rely on internal guidelines for the selection of open source products. Our experience has shown that in most cases project leaders select an open source solution based on its being readily available and fulfilling their functional requirements [5, 6]. FOCSE evaluation criteria are aimed at evaluating each open source project in its entirety, highlighting the promptness of reacting against newly discovered vulnerabilities or incidents. Applications success, in fact, depends on the above principle because a low reaction rate to new vulnerabilities or incidents implies higher risk for users that adopt the software, potentially causing loss of information and money.

3.1 Evaluation principles

FOCSE evaluation is based on six macro-areas [5]:

- *Generic Aspects (GA)*. i.e. all quantitative attributes expressing the solution's non-functional features, i.e. those not related to its purpose or scope (for a complete list, see [8]).
- *Developers Community (DC)*. i.e. quantitative attributes expressing the composition and diversity of the developers community. A high number of developers from different countries allows sharing of diverse backgrounds and skills, giving vitality and freshness to the community and helping in solving problems, including bugs definition and fixing.
- *Users Community (UC)*. The success of an open source application can be measured in terms of number and profile of the users that adopt it and rely on it. Obviously, measuring and evaluating the users community is less simple than doing so for developers because users interacting with an open source project are often anonymous. The users community, however, can be quantitatively estimated by means of parameters like the number of downloads, the number of requests, the number of posts inside the forum, and the number of users subscribed to the mailing list. A qualitative measure of this macro-area can also be given by the profile of the users adopting the project: if users belong to well-known companies or organizations and report positive results, the solution's UC indicators can be enhanced.

[1] Of course, FOCSE does not mandate the use of FLOSSmole, as data can be gathered manually by the evaluator. However, companies interested in comparative evaluation of OSS solutions should rely on a certified and repeatable data collection technique.

- *Software Quality (SQ).* This area include metrics of quality built into the software by the requirements, design, code and verification processes to ensure that reliability, maintainability, and other quality factors are met.[2]
- *Documentation and Interaction support (DIS).* This macro area is composed of two major sub-areas: *traditional documentation,* that explains the characteristics, functionalities and peculiarities of the software and *support,* in terms of time allotted by developers to give feedback via forums, mailing lists, white papers, and presentations.
- *Integration and Adaptability with new and existing technologies (IA).* A fundamental tenet of OS projects is full integration with existing technologies at project startup and a high level of adaptability to new technologies presented during project life. Another fundamental aspect is the ability of the developers' community to solve and fix bugs and react to new vulnerabilities.

3.2 Evaluation parameters

We now provide the detailed description of some metrics (see Table 1 and 2) and their distribution in the six macro-areas described above. These metrics are then used (Section 5) to evaluate and compare open source security applications. Regardless of the macro-area they belong to, our quantitative metrics are orthogonally divided in: *i) Core Metrics (CM)*, and *ii) Advanced Metrics (AM).*

Core Metrics

Core Metrics include all metrics that can be readily computed from current information on the projects. These metrics are based on data that can be usually found in the projects web sites; however structured data sources like FLOSSmole [13] can make the evaluation process stronger and more trustworthy.

- *Age*, that represents the age in days of the project, calculated from the date of the first stable release.
- *Last Update Age*, that represents the age in days of the last stable project update. It is calculated as the difference between the date of the last update and the current date. Differently from *Age* metric, *Last Update Age* measures the level of freshness of the last application update, and it allows the identification of dead projects.
- *Project Core Group*, a boolean value that evaluates the existence of a stable core group of developers who have been working on the project from its inception (or for at least three-quarters of its Age). Core developers are

[2] Open source security solutions lend themselves to quality assurance and evaluation based on shared testing and code walk-through as outlined in [1]. However, comparing reference implementations of security solutions based on code walk-through is outside the scope of this paper.

defined as the ones that contribute both to project management and code implementation.

- *Number of Core Developers*, strictly related to the above Project Core Group metric, measures the number of core developers.
- *Number of Releases*, that measures the number of releases from the project start up. A high number of releases could indicate the vitality of the community and its promptness on reacting against new threats and vulnerabilities.

Table 1. Evaluation Metrics Definition: Core Metrics

Core Metrics			
Name	**Definition**	**Values**	**Area**
Age	Age of the project.	Days	GA
Last Update Age	Age of the last project update.	Days	GA
Project Core Group	Evaluate the existence of a group of core developers.	Boolean	GA,DC
Number of Core Developers	Number of core developers contributing the project.	Integer	DC
Number of Releases	Number of releases since project start up.	Integer	SQ,IA
Bug Fixing Rate	Rate of bugs fixed.	Real	SQ,IA
Update Average Time	Vitality of developers group, i.e. mean number of days to wait for a new update (release or patch).	Days	SQ,IA
Forum and Mailing List Support	Check forum and mailing list availability.	Boolean	GA,DIS
RSS Support	Check RSS availability.	Days	GA,DIS
Number of Users	Number of users adopting the application.	Integer	UC
Documentation Level	Level of project documentation, in terms of API, user manuals, whitepapers.	Mbyte	DIS
Code Quality	Qualitative measure of code quality.		SQ,IA
Community Vitality	Vitality of the community in terms of number of forum threads and replies.	Real	DC,UC

- *Bug Fixing Rate*, which measures the rate of bugs fixed looking at bugs and fixings reports of each product. To prevent young projects with few bugs fixed from outperforming old and stable projects with hundreds of bugs fixed, the bug fixing rate is weighted over the total number of bugs detected. This rate is computed as:

  $$\left(\#ofBugsFixed + 1/\#ofBugsDetected + 1\right)\left(1 - e^{-\#ofBugsDetected}\right).$$

  We stress the fact that this metrics is available from well known security-related sources, such as the Computer Emergency Response Team (CERT) [9], providing detailed information about discovered bugs.

- *Update Average Time*, measuring the vitality of an open source community. It indicates the average number of days between releases of major and minor versions (patches) of the product. This metric is calculated as: *age/(#ofPatches + #ofReleases)*.
- *Forum and Mailing List Support*, a boolean value expressing availability of forum and mailing lists at the products' web sites. This is an important feature of open source products since it hints at a strict collaboration between users and developers communities.
- *RSS Support*, a boolean value expressing availability of RSS (Really Simple Syndication), i.e., a family of web feed formats, used to publish frequently updated digital content. This is an important feature of open source products since it allows users who download and rely on a particular project to be fully informed of the project news.
- *Number of Users*, expressing the number of users adopting the product; this value can be roughly approximated as: *#ofDownloads/#ofReleases*. This parameter is also an indicator of the product's popularity.
- *Documentation Level*, expressing the documentation level (in Mbyte) in terms of APIs documentation, user manuals, whitepapers, and so on.
- *Code Quality*, that measures the intrinsic quality of the software product.[3]
- *Community Vitality*, that measures the vitality of the community in term of answers given in the forum in response to specific users questions. This value is computed as: *#ofForumReplies/#ofForumThreads*.

The core metrics set is summarized in Table 1.

Advanced Metrics

Advanced Metrics include evaluation parameters requiring privileged access to the developers' community [5]. Otherwise, they can be estimated basing on raw data**.**

- *Group/Developers Stability*, that measures the degree of stability of developers group and, consequently, the stability of the product itself. Each developer is classified as *stable* or *transient*, where stable is a developer that continuously contributes to code, transient in the other cases. The exact number of contributions to make a developer stable is project-dependent. This value is computed as: *#ofStableDevelopers/#ofDevelopers*100*.
- *Project Reputation*, which estimates the reputation of the project by aggregating the evaluations provided by the project developers and users. Several algorithms for assessing reputation are available [16].
- *Repository Quality*, that provides an estimation of the repository where project is hosted.[4] It can be computed in several different ways; we chose to

[3] This metric is included for completeness, but its measurement is out of the scope of this paper (see Section 3.1).

[4] For self-hosted projects this metrics is set to 0.

measure it as the number of active projects hosted by the repository (an active project is defined as one that has released at least an update within a year), over the total number of hosted projects: *#ofActiveProjects/#ofProjects*.

- *Reaction Rate*, that estimates the average time the developers community takes to find solutions to newly discovered vulnerabilities. This parameter measures the community's promptness in terms of reaction against discovered software vulnerabilities.
  Given V as the set of vulnerabilities, this metric is defined as:

$$\left(n * UpdateAverageTime\right) \Big/ \sum_{i=1}^{n} FixingDate\left(V_i\right) - DiscoveringDate\left(V_i\right)$$

  where $V_i \in V$ and $n = |V|$.

- *Incident Frequency*, that measures the robustness of the application with respect to newly discovered vulnerabilities. This parameter is computed as *#ofIncidents/|V|* where *V* is the set of vulnerabilities.

Table 2. Evaluation Metrics Definition: Advanced Metrics

Advanced Metrics			
Name	**Definition**	**Values**	**Area**
Group/Developers Stability	Measures the degree of stability of a developers group.	[0..100%]	DC
Reaction Rate	Average time needed by the developers' community to find solutions for newly discovered vulnerabilities. More specifically, it represents the project developers' ability in reacting to the set of vulnerabilities.		IA
Repository Quality	Estimator of the project repository quality.		GA
Incident Frequency	Measures the number of incidents due to vulnerabilities.		IA
Project Reputation	Measure the project reputation by aggregating the evaluation provided by project developers and users.		GA

The first metrics, Groups/Developers Stability, is not easy to estimate from outside the developers' community. It may be however available to insiders, e.g. to companies that adopt an open source product and actively contribute to its community. Finally, regarding the computation of the last two parameters, we stress the fact that various security-related Web portals provide databases that contain information about vulnerabilities and related incidents summaries. In particular, three main portals stand out: *Secunia* (http://secunia.com/) that offers monitoring of vulnerabilities in more than 12,000 products, *Open Source*

Vulnerability Database (OSVDB) (http://www.osvdb.org/) an independent database that provides technical information about vulnerabilities and, finally, CERT, which provides a database containing information about vulnerabilities, incidents and fixes.

In summary, most of the information required to compute FOCSE advanced metrics is already available on the Net. Unfortunately, this information being in raw format makes it difficult to automate the computation, as substantial pre-processing is needed to compute these metrics.

4 Aggregating Heterogeneous Results

To generate a single estimation, it is necessary to aggregate the metrics values. This way, two or more projects, each one described by its set of attributes, can be ranked by looking at their FOCSE estimations. Below, the *Ordered Weighted Average* (OWA) operator, used to aggregate the defined metrics, is introduced.

4.1 OWA Operator

Ordered Weighted Averaging (OWA) operators, originally introduced by Ronald Yager [24, 26], provide a parameterized family of mean-type aggregation operators. An important feature of these operators is the *reordering* step, which makes OWA a nonlinear operator. OWA operator is different from the classical weighted average in that coefficients are not associated directly with a particular attribute but rather to an ordered position. The structure of these operators is aimed at combining the criteria under the guidance of a quantifier.

Definition 1 (OWA Operator) *Let* $w = [\omega_1, \omega_2, \ldots, \omega_n]$ *a weight vector of dimension* n*, such that* $\omega_i \in [0,1]$ *and* $\sum_i \omega_i = 1$. *A mapping* $f_{OWA} : R^n \longrightarrow R$ *is an OWA operator of dimension* n *if*

$$f_{OWA}(a_1, a_2, \ldots, a_n) = \sum_i \omega_i a_{\sigma(i)}$$

where $\{\sigma(1), \ldots, \sigma(n)\}$ *is a permutation of {1,...,n} such that* $a_{\sigma(i)} \leq a_{\sigma(i-1)}$ *for i=2,…,n.*

We adopt the monotonic quantifiers Q_{mean} [26]. The pure averaging quantifier has w_j=1/n for all j=1,…,n having Q_{mean}(K)=K/n as its linear quantifier.

The previous quantifier represents the set of weights used in our experimentation (i.e., [1/n, 2/n,…,(n-1)/n, 1]). All decision process involving multiple criteria like

software selection involve some compensatory trade-offs. Trade-offs occurs in the presence of conflicting goals, when compensation between the corresponding compatibilities is allowed. OWA operators can realize trade-offs between objectives, by allowing a positive compensation between ratings, i.e. a higher degree of satisfaction of one of the criteria can compensate for a lower degree of satisfaction of other criteria to a certain extent. OWA operators provide for any level of compensation lying between logical conjunction and disjunction. An interesting feature of OWAs is their adaptability. To any specific software selection problem we can tailor an appropriate OWA aggregation operator from some rules and/or samples determined by the decision makers.

5 Applying FOCSE to Existing Critical Application

We introduce two categories of open source security solutions: SSO systems and SSH clients. Then, we show how selection can be made by first evaluating the FOCSE metrics, and then by aggregating them by means of OWA operator.

5.1 Single Sign-On Frameworks

The SSO [15] approach is aimed at co-ordinating and integrating user log-on mechanisms of different domains. In particular, SSO provides a technique where a primary domain is responsible for managing all user credentials and information used during the authentication process, both to the primary domain itself and to each of the secondary domains that the user may potentially require to interact with. SSO also provides the users with a transparent authentication to the secondary domains. In this scenario, the following subset of SSO frameworks has been evaluated by FOCSE metrics (for more details, see [2]).

- *Central Authentication Service*. Central Authentication Service (CAS) [7] is an open source authentication system originally developed at Yale University. It implements a SSO mechanism aimed at providing a *Centralized Authentication* to a single server through *HTTP redirections*.
- *SourceID*. SourceID [22], first released in 2001 by Ping Identity Corporation, is an open source multi-protocol project for enabling identity federation and cross-boundary security. SourceID also implements Liberty Alliance SSO specifications.
- *Java Open Single Sign-On (JOSSO)*. Java Open Single Sign-On (JOSSO) is an open source SSO infrastructure based on J2EE specifications. In detail, JOSSO provides a solution for centralized platform-neutral user authentication [17], combining several authentication schemes (e.g., username/password or certificate-based) and credential stores.
- *Open Source Web SSO*. The Open Source Web SSO (Open SSO) [18] project relies on the consolidated Web SSO framework developed by Sun

Microsystems, that was opened to the open source community in July 2005. It provides services and functionalities for implementing transparent SSO as an infrastructure security component.

5.2 SSH Clients

SSH is a communication protocol widely adopted in the Internet environment that provides important services like secure login connections, file transfers and secure forwarding of X11 connections [27]. SSH protocol allows also a communication approach named Tunneling as a way to encapsulate a generic communication flow in SSH packets, implementing a port forwarding mechanism and securing data that use untrusted communication protocols, exploiting SSH encryption features. The FOCSE framework has been applied for evaluating the following SSH clients.

- *Putty*. Putty [20] is a popular open source SSH client for Microsoft Windows platforms. It supports versions 1 and 2 of SSH protocol, terminal emulation, and provides a complete and essential user interface.
- *WinSCP*. WinSCP [25] allows safe copying of files between remote internet machines through the SSH protocol. It also offers basic remote management operations, such as file duplication, renaming and deleting, and supports all the encryption features of SSH protocol.
- *ClusterSSH*. ClusterSSH [23] allows users to open and manage several terminal windows with connections to specified hosts and an administration console. The tool is also intended for cluster administration.

5.3 SSO Comparison

Table 3 gives a comparison of SSO implementations based on FOCSE metrics.[5] Focusing on evaluation, as shown by Table 3, all systems are quite stable due to the fact that their start-up happened more than a year ago. Even Open SSO, i.e. the most recent one, can be considered as a stable implementation since it represents an open source extension of a well-established proprietary implementation, named *Sun Java System Access Manager*. A common characteristic shared by all analyzed solutions is that they are managed by a consolidated core group providing stability to the project and coordination to open source community. By contrast, these solutions have different documentation levels. Specifically, whereas CAS provides a good amount of documentation, i.e. `28.55 MB`, SourceID presents on its Web sites only a limited amount of information, i.e. `8.96 MB`. Although at the first sight the number of releases could seem a good estimation of projects vitality, this is not entirely true. Often, in fact, the number of releases is highly dependent on the project age. To the

[5] Due to the fact that only JOSSO and Open SSO data have been gathered by FLOSSmole, our evaluation is sensitive to errors due to obsolete information.

purpose of clearly evaluating the liveness of a project, the number of releases should be coupled with the Update Average Time. In particular, Table 3 seems to suggest that Open SSO is the liveliest project. However, its low update average time is due to the fixing of youth problems that hints to keep Open SSO out of this metric comparison. We argue, then, that the more active and viable implementation is JOSSO, because it provides a new release every `44 days`. Also, the Bug Fixing Rate metric suggests that JOSSO is the most reactive project between the analyzed solutions. Concerning *Last Update Time*, CAS implementation achieves the best results, i.e. `18 days`. Also, CAS is the only project providing a certified list of users. Here we do not consider the Number of Users, Repository Quality, and Community Vitality parameters for all solutions, because relatively few solutions provide enough information to clearly and unambiguously compute them. In particular, CAS provides a certified list of the CAS' deployers, and JOSSO and Open SSO make possible to easily compute the community vitality.

Table 3. Comparison of proposed SSO implementations at 31 December 2006.

Metrics	CAS	SourceID	JOSSO	Open SSO
Age (GA)	1865 days	1177 days	854 days	570 days
Last Update Age (GA)	18 days	236 days	217 days	21 days
Project Core Group (GA,DC)	Yes	Yes	Yes	Yes
Number of Core Developers (DC)	5	N/A	2	N/A
Number of Releases (SQ,IA)	28	7	7	1 (since code opening)
Bug Fixing Rate (SQ,IA)	N/A	N/A	0.78	0.53
Update Average Time (SQ,IA)	67 days	168 days	44 days	27 days
Forum and Mailing List Support (GA,DIS)	Mailing List Only	Mailing List Only	Yes	Yes
RSS Support (GA,DIS)	Yes	Yes	No	Yes
Number of Users (UC)	48 (certified)	N/A	7072 (approx.)	N/A
Documentation Level (DIS)	28.55 MB	8.96 MB	16.96 MB	14.3 MB
Community Vitality (DC,UC)	N/A	N/A	1.87	3.56

5.4 SSH Comparison

Table 4 gives a comparison of SSH client implementations. Differently from SSO systems, all the analyzed SSH frameworks lie in FLOSSmole database.

Table 4. Comparison of proposed SSH implementations at 31 December 2006.

Metrics	Putty	SourceID	JOSSO
Age (GA)	2911 days	1470 days	1582 days
Last Update Age (GA)	636 days	238 days	159 days
Project Core Group (GA,DC)	Yes	Yes	Yes

Number of Core Developers (DC)	4	2	2
Number of Releases (SQ,IA)	15	32	15
Bug Fixing Rate (SQ,IA)	0.67	N/A	0.85
Update Average Time (SQ,IA)	194 days	46 days	105 days
Forum and Mailing List Support (GA,DIS)	N/A	Forum Only	Yes
RSS Support (GA,DIS)	No	Yes	Yes
Number of Users (UC)	N/A	344k	927
Documentation Level (DIS)	1.39 MB	10 MB	N/A
Community Vitality (DC,UC)	N/A	3.73	5.72

Focusing on the evaluation, it is clear that all the projects are stable since their startup happens more than four years ago. This results in stable and consolidated project core groups of at least two core developers, and in a good number of releases. Concerning the *Bug Fixing Rate* metric, whereas for WinSCP no data are available, Putty and ClusterSSH provide a good bug fixing rate, 0.67 and 0.85 respectively.

To conclude, the number of users adopting WinSCP (i.e., 344K of users) is impressive, suggesting that it is very attractive for users to take advantage of open source SSH solutions.

5.5 Applying OWA Operator to FOCSE Critical Application Comparison

We now apply OWA operator to provide a single estimation of each evaluated solution. For the sake of conciseness, we shall only show the details of OWA application to CAS solution. All other solutions can be processed in the same vein.

First, the adoption of OWA operator together with the Q_{mean} quantifier results in the following set of weights:

$$w=[\tfrac{1}{12},\tfrac{2}{12},\tfrac{3}{12},\tfrac{4}{12},\tfrac{5}{12},\tfrac{6}{12},\tfrac{7}{12},\tfrac{8}{12},\tfrac{9}{12},\tfrac{10}{12},\tfrac{11}{12},1]$$

In particular, the Q_{mean} identifier is used to mitigate the impact of too high and too low values on the overall aggregation process.[6] Then, after normalizing the vector of weights:

$$w_n=w/\tfrac{78}{12}=[\tfrac{1}{78},\tfrac{1}{39},\tfrac{1}{26},\tfrac{2}{39},\tfrac{5}{78},\tfrac{1}{13},\tfrac{7}{78},\tfrac{4}{39},\tfrac{3}{26},\tfrac{5}{39},\tfrac{11}{78},\tfrac{2}{13}]$$

we normalized the vector of CAS attributes as $attrValue_k/maxAttrValue$, where $attrValue_k$ is the value of the k-th attribute and $maxAttrValue$ is the maximum attribute value among all the analyzed solutions. It is important to underline that attributes such as *Last Update Age*, where low values mean better evaluations, are normalized as 1- $attrValue_k/maxAttrValue$. The normalization process results in the ordered vector a=[1, 1, 1, 1, 1, 1, 0.93, 0.5, 0.4, 0.01, 0, 0].

Now, we calculate the final value of CAS system as: $f_{OWA}(a)=\sum_{i=1}^{12} w_i a_i = 0.45$.

[6] Different quantifiers could be adopted depending on the scenario.

When the same process is applied to all solutions, one obtains the two tables depicted in Table 5. To conclude, from Table 5 is clear that whereas among SSO systems the best solution is CAS, followed by JOSSO implementation, concerning SSH clients, the solution more likely to be adopted is WinSCP.

Table 5. OWA-based Comparison.

(a) SSO Comparison

	CAS	**SourceID**	**JOSSO**	**Open SSO**
f_{OWA}	0.45	0.19	0.36	0.34

(b) SSH Comparison

	Putty	**SourceID**	**JOSSO**
f_{OWA}	0.23	0.51	0.47

6 Conclusions and Future Work

We presented our FOCSE framework aimed at the definition of a quantitative approach to the comparative evaluation of security-related open source systems. A structured set of metrics used in the evaluation process and specifically designed for such systems, a formal aggregation is introduced to deal with the heterogeneity of such metrics. This aggregation allows the FOCSE evaluation to be expressed by means of a single value and to be more user-friendly. Then as case-studies, we compared some well-known implementations of SSO and SSH applications. Future work will study the integration of FLOSSmole-like databases in FOCSE, allowing the definition of an infrastructure able to gather the requested data by itself and then provide the evaluation in a transparent way to the user. Also the definition of a validation system of open source projects based on community inputs [19], as well as the definition of an extended version of the framework able to evaluate whatever open source solution will be subject of future research.

Acknowledgements

This work was supported in part by the European Union within the PRIME Project in the FP6/IST Programme under contract IST-2002-507591, and by the by the Italian Ministry of Research under FIRB contracts n. RBNE05FKZ2_004 TEKNE and n. RBNE01JRK8_003 MAPS.

References

1. S. Abiteboul, X. Leroy, B. Vrdoljak, R. Di Cosmo, S. Fermigier, S. Lauriere, F. Lepied, R. Pop, F. Villard, J.P. Smets, C. Bryce, K.R. Dittrich, T. Milo, A. Sagi, Y. Shtossel, and E.

Panto. Edos: Environment for the development and distribution of open source software. In *Proc of The First International Conference on Open Source Systems*, pages 66–70, Genova (Italy), July 2005.

2. C.A. Ardagna, E. Damiani, S. De Capitani di Vimercati, F. Frati, and P. Samarati. CAS++: an open source single sign-on solution for secure e-services. In *Proc. of the 21st IFIP International Information Security Conference "Security and Privacy in Dynamic Environments"*, May 2006.
3. C.A. Ardagna, E. Damiani, F. Frati, and M. Madravio. Open source solution to secure e-government services. *Encyclopedia of Digital Government*, 2006.
4. C.A. Ardagna, E. Damiani, F. Frati, and M. Montel. Using open source middleware for securing e-gov applications. *In Proc. of The First International Conference on Open Source Systems*, pages 172–178, Genova (Italy), July 2005.
5. C.A. Ardagna, E. Damiani, F. Frati, and S. Reale. Adopting open source for mission-critical applications: A case study on single sign-on. In *Proc. of IFIP Working Group 2.13 Foundation on Open Source Software*, volume 203/2006, pages 209–220, Como, Italy, 2006.
6. C.A. Ardagna, E. Damiani, F. Frati, and S. Reale. Secure authentication process for high sensitive data e-services: A roadmap. *Journal of Cases on Information Technology*, 9(1):20–35, 2007.
7. P. Aubry, V. Mathieu, and J. Marchal. Esup-portal: open source single sign-on with cas (central authentication service*). In Proc. of EUNIS04 – IT Innovation in a Changing World*, pages 172–178, Bled (Slovenia), 2005.
8. A. Capiluppi, P. Lago, and M. Morisio. Characteristics of open source projects. In *CSMR*, page 317, 2003.
9. CERT-CC. Cert coordination center. Available: www.cert.org/.
10. M. Conklin. Beyond low-hanging fruit: Seeking the next generation in floss data mining. In *Proc. of IFIP Working Group 2.13 Foundation on Open Source Software*, volume 203/2006, Como, Italy, 2006.
11. C. Cowan. Software security for open-source systems. *IEEE-SEC-PRIV*, 1(1):38–45, January/February 2003.
12. J. Feller and B. Fitzgerald. A framework analysis of the open source software development paradigm. In *ICIS*, pages 58–69, 2000.
13. FLOSSmole. Collaborative collection and analysis of free/libre/open source project data. Available: ossmole.sourceforge.net/.
14. B. Golden. Succeeding with Open Source. *Addison-Wesley Professional*, 2004.
15. The Open Group. Single sign-on. Available: www.opengroup.org/security/sso/.
16. A. Josang, R. Ismail, and C. Boyd. A survey of trust and reputation systems for online service provision. In *Decision Support Systems*, 2005.
17. JOSSO. Java open single sign-on. Available: sourceforge.net/projects/josso.
18. OpenSSO. Open web SSO. Available: opensso.dev.java.net/.
19. E. Damiani P. Ceravolo and M. Viviani. Bottom-up extraction and trust-based refinement of ontology metadata. *IEEE Transaction on Knowledge and Data Engineering*, 19(2):149–163, February 2007.
20. PuTTY. A free telnet/ssh client. Available: www.chiark.greenend.org.uk/~sgtatham/putty/.
21. E.S. Raymond. The cathedral and the bazaar. Available: www.openresources.com/documents/cathedral-bazaar/, August 1998.
22. SourceID. Open source federated identity management. Available: www.sourceid.org/.
23. Cluster SSH. Cluster admin via ssh. Available: sourceforge.net/projects/clusterssh.
24. V. Torra. The weighted OWA operator. *International Journal of Intelligent Systems*, 12(2):153–166, 1997.
25. WinSCP. Free sftp and scp client for windows. Available: winscp.net/eng/index.php.
26. R.R. Yager. On ordered weighted averaging aggregation operators in multi-criteria decision making. *IEEE Transaction Systems, Man, Cybernetics*, 18(1):183–190, January/February 1988.
27. T. Ylonen. Ssh - secure login connections over the internet. In *Proc. of Sixth USENIX Security Symposium*, page 3742, San Jose, California, USA, 1996.