Proceedings of the 52nd Hawaii International Conference on System Sciences | 2019

Data-driven Selection of Security Application Frameworks During Architectural Design

Humberto Cervantes
Universidad Autónoma Metropolitana - Iztapalapa
hcm@xanum.uam.mx

Rick Kazman
University of Hawaii at Manoa
kazman@hawaii.edu

Jungwoo Ryoo
Pennsylvania State University at Altoona
jxr65@psu.edu

Junsung Cho, Geumhwan Cho, Hyoungshick Kim
Sungkyunkwan University
js.cho@skku.edu, geumhwan@skku.edu, hyoung@skku.edu

Jina Kang
National Security Research Institute
4558kang@nsr.re.kr

Abstract

The selection of application frameworks is an important aspect of architectural design. Selection often requires satisficing, that is, searching a potentially large space of design alternatives until an acceptable solution is found. There is, however, little help for architects in selecting software frameworks. In this paper we investigate the criteria used by practicing software architects in selecting security frameworks. We also propose how information associated with some of the criteria that are important to architects can be obtained manually or in an automated way from online sources such as GitHub. Our ultimate goal is to identify measures associated with these criteria that can be helpful in providing support for architects to select software frameworks.

1. Introduction

Software architecture design is the activity of making design decisions to identify elements that compose the architecture of a software system. Architectural design can be performed systematically using a method such as Attribute-Driven Design (ADD) [3]. Generally, the design of a software architecture is performed by reusing proven solutions to recurring design problems. These proven solutions can be conceptual in nature, such as design patterns [4], or they can be concrete, such as application frameworks. An application framework (or just framework) is a collection of reusable software elements that provide generic functionality addressing recurring domain and quality attribute concerns across a broad range of applications. There exist application frameworks for many problem domains including user interfaces, object-oriented to relational mapping (ORM), generation of reports, and security. An example of an application framework for ORM is hibernate[1]. Using this framework involves including several library (Jar) files along with annotations in certain classes so that the framework can successfully persist these classes in a relational database. Other Java frameworks follow a similar usage pattern.

Architects generally select frameworks as part of the design process, and selecting an appropriate framework is an important design decision which may be costly to revert. The goal of our research is to help software architects make better and more informed design decisions, particularly regarding the selection of application frameworks during architectural design. In this paper we identify a list of criteria that can be used in the selection of software security frameworks. Using this list of criteria we surveyed practicing software architects to understand how important these criteria are to them. We then investigate how data associated with a subset of these criteria can be obtained from online sources such as GitHub to provide relevant information to the architects.

2. Context of the study

[1] http://hibernate.org/

URI: https://hdl.handle.net/10125/60170
ISBN: 978-0-9981331-2-6

In this section we discuss the evaluation criteria we chose, and the frameworks we selected for our study.

2.1. Criteria

Application frameworks can be categorized according to different characteristics, or criteria, that are taken into account for their selection. We based this list of criteria on informal interviews with software architects, and later validated it with a survey, as we will describe in Section 3. Table 1 lists the criteria we selected and the meaning of each one of them.

The criteria are listed in order of importance, as determined by our survey respondents (see http://sites.psu.edu/frameworks/survey/). But it should be noted that the differences in scores from top to bottom were not large: the most important criterion was just 25% more highly ranked than the least important.

Table 1. Selection criteria

Criterion	Meaning
Functional completeness	The framework offers the functions that are needed
Ease of integration	The framework integrates easily with other technologies that are used in the project.
Community engagement	How quickly bugs are fixed
Quality of documentation	Good documentation and examples are available
Cost	The framework is free or reasonably priced
Usability	The framework is easy to use
Learnability	The framework is easy to learn
Support	The vendor or community provides support, answers questions quickly
Familiarity	I or my team were already familiar with it
Popularity	Lots of other projects are already using this framework
Run-time performance	The framework does not introduce an unacceptable performance penalty
Evolution	New, useful features are regularly added to the framework

2.2. Selected frameworks

In this study we have chosen to focus on open source software security frameworks supporting the Java programming language and dedicated to the quality attribute of security. This is because the domain of security frameworks is rich, with many products available, and Java has been the most popular programming language for enterprise applications for well over a decade[2]. Thus our chosen domain is a rich one. And we chose the most popular security frameworks within this chosen domain. But this selection was simply to scope our analysis efforts. Note that none of the techniques that we apply are specific to either the Java language or to the domain of security.

Table 2. Considered security frameworks

ID	Name and version	Focus	Description
F1	Spring Security[3] (v 4.1.0)	A&A	Highly customizable authentication and access-control framework.
F2	Bouncycastle[4] (v 1.54)	CRY	Java cryptography APIs.
F3	JAAS[5] (Java SE 8)	A&A	The Java Authentication and Authorization Service is part of the Java security APIs.
F4	JCE[6] (Java SE 8)	CRY	The Java Cryptography Extension is part of the Java security APIs.
F5	Apache Shiro[7] (v 1.2.4)	A&A	Powerful and easy-to-use Java security framework that performs authentication, authorization, cryptography, and session management.
F6	Jasypt[8] (v 1.9.2)	CRY	Java library which allows the developer to add basic

[2] https://www.tiobe.com/tiobe-index/
[3] http://projects.spring.io/spring-security/
[4] http://www.bouncycastle.org/
[5] http://www.oracle.com/technetwork/java/javase/jaas
[6] http://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html
[7] http://shiro.apache.org/

			encryption capabilities to his/her projects with minimum effort.
F7	HDIV[9] (v 2.1.11)	WEB	Open-source Java web application security framework that eliminates or mitigates web security risks by design for some of the most used JVM web frameworks.
F8	OWASP ESAPI[10] (v 2.1.0)	WEB	This is the Java EE language version of OWASP ESAPI (Enterprise Security API).
F9	Keyczar[11] (v 0.71)	CRY	Open source cryptographic toolkit designed to make it easier and safer for developers to use cryptography in their applications.
F10	OACC[12] (v 2.0.0)	A&A	OACC provides a high performance API that provides permission-based authorization services for Java applications.
F11	jGuard[13] (v 1.0.4)	A&A	jGuard is a library that provides easy security (authentication and authorization) for Java web applications.
F12	PicketLink[14] (v 2.7.1)	A&A	PicketLink provides an alternative to JEE Security, providing a rich, powerful and flexible API to secure your applications.

We include standalone security frameworks and APIs that are part of full stack frameworks (such as JAAS). We exclude non-security frameworks (such as presentation layer frameworks like ZK) which, while they may include security functionality, do not primarily focus on security. We constrain our scope in this way to avoid making "apples and oranges" comparisons; that is, to avoid comparing frameworks with fundamentally different objectives. Table 2 lists the frameworks that we considered for this paper. It should be noted that the majority of these frameworks have a relatively narrow focus, as described in the frameworks' homepages. This focus is typically either authorization and authentication (A&A - 6 frameworks), cryptography (CRY - 4 frameworks), or protection of web applications against attacks (WEB - 2 frameworks). The descriptions in table 2 come from the frameworks' homepages.

[8] http://www.jasypt.org/

[9] http://www.hdiv.org/

[10] https://github.com/ESAPI/esapi-java-legacy

[11] https://github.com/google/keyczar

[12] http://oaccframework.org/

[13] http://jguard.xwiki.com

[14] http://picketlink.org/appsecurity/

3. Survey of practicing architects

To understand the criteria used by architects in selecting security frameworks, we created a survey which can be found at the following URL (https://goo.gl/forms/bDT3b9yNkp4FxxCw1). The survey consists of questions regarding:

1. The participant's background (years of experience in total and as architect, years of experience with Java frameworks)
2. Which of the frameworks from Table 2 the architect is familiar with, and whether he or she was responsible for selecting the framework in a project
3. Framework-specific questions for up to five frameworks, including which of the criteria in table 1 were taken into account when a particular framework was selected (these were multiple answer questions with answers ranging from "primary reason for selection" to "did not take into account").
4. How each one of the criteria in table 1 is evaluated (this was a free text response)

We received 14 responses to our survey. 85.6% of the respondents had 10 or more years of experience in software development. 78.6% had 4 or more years as an architect or technical lead, all of the respondents had 5 or more years of experience using Java Frameworks. 80% of the respondents had been responsible for the selection of a security framework in a project where they worked.

The frameworks that were more popular among the respondents were the following (the percentage indicates the relative number of participants that claimed they were familiar with it):

- Spring Security (86.7%)
- JAAS (73.3%)
- JCE (66.7%)
- BouncyCastle (33.3%)

Due to limited space, we only provide highlights of the results from the survey but the complete results can be

reviewed at the following address http://sites.psu.edu/frameworks/survey/. Of these frameworks, Spring Security was overwhelmingly chosen by the respondents in 11 of the 14 responses. Other architects also selected JCE, JAAS and Bouncastle but these frameworks were only selected once each.

4. Data collection

In this section we present data that is collected for 4 of the criteria listed previously: functional completeness, community engagement, evolution and popularity. These criteria were selected to explore different strategies of data collection from online sources.

4.1. Functional completeness

We measure functional completeness in terms of coverage of the domain. For this reason we scrutinize each framework to determine how many distinct areas of security concerns the framework addresses. Security tactics exhaustively define the facets of software security that a framework could be architected to address. Tactics are generic design primitives that have been organized according to the quality attribute that they primarily affect: availability, modifiability, security, usability, testability, and so forth [2]. Tactics have been used to guide both design and analysis [3] [9]. The way that we employ tactics here is as a kind of analysis: tactics describe the space of possible design objectives with respect to a quality attribute, and by determining which tactics a framework realizes, we get a measure of the functional completeness of the framework. In this way we can rank the frameworks according to the degree of each framework's coverage of security tactics. Security tactics abstract the complete domain of design choices for software security. Figure 1 shows the security tactics hierarchy. There are four broad software-based strategies for addressing security: detecting, resisting, reacting to, and recovering from attacks. These are the top-level design choices that an architect can make when considering how to address software security. The leaf nodes further refine these top-level categories. For example, to resist an attack an architect may choose to authorize users, authenticate users, validate input, encrypt data, etc. Each of these is a separate design choice that must be implemented, either by custom coding or by employing a software component such as a framework.

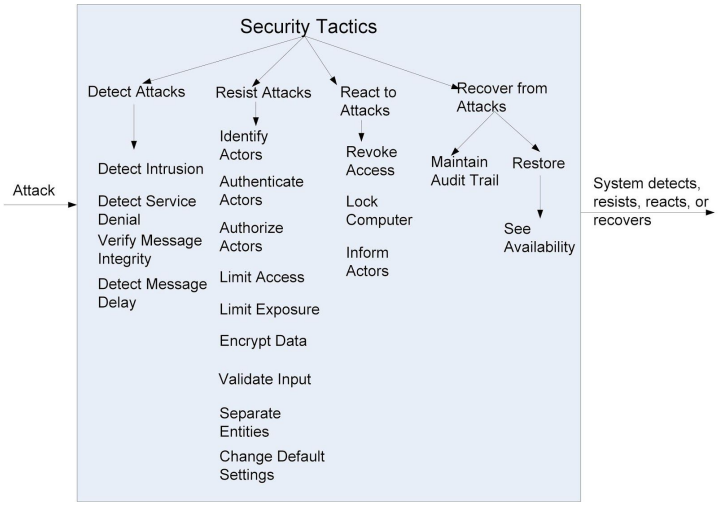

Figure 1: Security Tactics Hierarchy

By understanding which specific tactics are addressed by a security framework, we measure its functional coverage (reported as the number of tactics that are covered by the framework). For example, a security framework may specialize in providing encryption features and hence is only implementing the 'Encrypt Data' tactic. This means that the functional coverage of the framework is quite limited as it only covers a single tactic.

To measure functional coverage, we first reviewed the published descriptions of all the frameworks under investigation. These descriptions were primarily obtained from the frameworks' homepages. An example is the description from the

Spring Security framework: "Spring Security is a framework that focuses on providing both authentication and authorization to Java applications. Like all Spring projects, the real power of Spring Security is found in how easily it can be extended to meet custom requirements". We also looked for additional materials such as online articles and tutorials. This initial review gave us an idea of the overall emphasis of each framework in their coverage. We then delved into the individual Application Programming Interfaces (APIs) that support specific security tactics to verify the claims made in the frameworks' descriptions.

Table 3: Extract of the template used to capture horizontal coverage information

#	Tactics group	Tactics question	Supported ? Yes/No/Not sure	If you answered Yes/Not Sure, please describe the features of the framework, which support the tactic (provide links if necessary) or describe why you are not sure. Preferably use official documentation.	If you answered Yes/Not Sure, please list the packages in the framework API, which are associated with the tactic. Use official API documentation to fill this section.
6	Resisting attacks	Does the framework support the **authentication of actors** ? An example is ensuring that an actor (user or a remote computer) is actually who or what it purports to be.	Yes	Spring Security provides different authentication options: ● In Memory Authentication ● JDBC Authentication ● LDAP Authentication http://docs.spring.io/spring-security/site/docs/4.0.4.RELEASE/reference/html/jc.html#jc-authentication See also a list of technologies that can be integrated for authentication purposes: http://docs.spring.io/spring-security/site/docs/4.0.4.RELEASE/reference/html/introduction.html#what-is-acegi-security	org.springframework.security.authentication org.springframework.security.authentication.dao org.springframework.security.authentication.encoding org.springframework.security.authentication.event org.springframework.security.authentication.jaas ...

To ensure the consistency across our data collection, the information was captured using a template (see table 3). This template served as a checklist, by listing all the known security tactics. It also served as a questionnaire, eliciting information such as whether a specific tactic is handled by the framework of interest, the details of the APIs used to realize the tactic, and additional justifications for how a decision on a framework's support for the tactic was made.

Table 4 summarizes the functional coverage of all the security frameworks we considered in this study (Y's mean the tactic is covered by the framework, the numbers F1 to F12 correspond to the frameworks listed in table 2). Note that in the table we omitted the tactics not covered by any of the frameworks we reviewed. Thus the tactics that are covered are the following:

- T1 - Identify actors
- T2 - Authenticate actors
- T3 - Authorize actors
- T4 - Encrypt data
- T5 - Limit access
- T6 - Validate input
- T7 - Verify message integrity
- T8 - Detect intrusion
- T9 - Maintain audit trail

Table 4 reveals that there are three distinct groups of software security frameworks. The first group includes frameworks that focus on cryptography. This group includes Jasypt, Keyczar, JCE and Bouncy Castle. The second group includes frameworks that focus on authentication and authorization. This group includes JGuard, OACC, PicketLink and JAAS. Finally, the third group

includes frameworks that strive to provide a comprehensive set of security features including cryptography, authentication, authorization, and others in the security tactics hierarchy. This third group includes OWASP ESAPI, Spring Security and Apache Shiro. These three groups can be contrasted with the focus that was initially identified by the descriptions in the framework homepages (see Table 2).

Table 4. Functional Coverage

	Resist Attacks						Detect Attacks		Recover
	T1	T2	T3	T4	T5	T6	T7	T8	T9
F1	Y	Y	Y		Y				
F2	Y	Y	Y		Y				
F3	Y	Y	Y		Y				
F4	Y	Y	Y		Y				
F5				Y			Y		
F6				Y			Y		
F7				Y			Y		
F8				Y			Y		
F9	Y	Y	Y	Y	Y	Y	Y	Y	Y
F10	Y	Y	Y	Y	Y				
F11	Y	Y	Y	Y	Y				
F12						Y		Y	Y

4.2. Community Engagement

We propose three different measures to evaluate the community engagement for frameworks. The first measure is the ratio of resolved issues vs. open issues. Issues include reported bugs that need to be fixed, and feature enhancement requests. We consider that a high resolution ratio indicates that the community that develops the framework actively and works towards improving its quality. The second measure is the average resolution time (ART), that is the average time it takes for issues to be resolved. This measure is calculated by Tr(i)- Tp(i) where Tp(i) is a timestamp created when an issue i is posted, and Tr(i) is a timestamp when the issue i is resolved. A smaller ART indicates that the members of the community actively work towards quickly addressing issues and improving the quality of the framework. The third measure is the number of contributors (and committers). A high number of contributors also indicates that there is a vigorous community committed to the development of the framework.

It should be noted that the calculation of these measures require that the framework has a publicly accessible issue tracking system. We used the official GitHub API v3 (https://developer.github.com/v3/) to obtain the issue tracking data in the case of Spring Security, PicketLink, OACC, Keyczar, ESAPI and Hdiv, all of which are using GitHub to handle project issues. Bouncy Castle and Apache Shiro use JIRA (https://www.atlassian.com/software/jira), which is a proprietary issue tracking tool developed by Atlassian. JIRA allowed us to export project issues as an Excel file. The Other frameworks, jGuard and Jasypt, are using sourceforge.net (https://sourceforge.net/), and we simply collected the necessary data manually. Of all the frameworks considered, only JAAS and JCE do not have publicly accessible issue tracking systems as they are developed as part of the Java API.

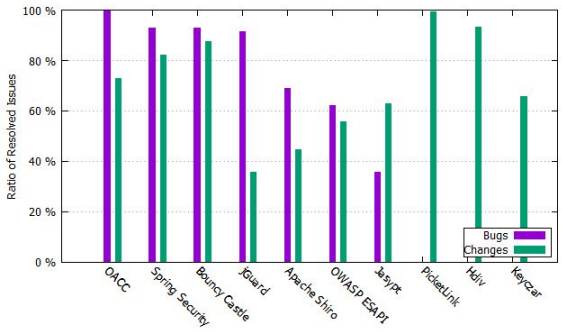

Fig 2: Ratio of resolved issues

Figure 2 shows the ratios of resolved issues including bug fixes and change requests (new features) for each of the frameworks evaluated. Here, "resolved" means a developer has either fixed the reported problem or satisfied the change request in a framework. We can see that PicketLink (99.20%) is the highest ranked security framework in terms of the ratio of resolved change requests. Moreover, Hdiv (93.33%), OACC (100% bug fixes and 72.73% changes), JGuard (91.49% bug fixes and 35.82% changes), Bouncy Castle (92.81 bug fixes and 87.37% changes), and Spring Security (92.92 bug fixes and 82.04% changes) are significantly better than the other frameworks in their ratio of resolved issues. Note that PicketLink, Hdiv, and Keyczar do not offer ways to distinguish bug fixes from other

types of change requests in their issue tracking system, which is why we classify them as generic changes (bars shown in green).

Figure "Distribution of issue resolution time for the security frameworks" on the accompanying website (https://sites.psu.edu/frameworks/) shows the distribution of the percentage of issues resolved across time periods (in days) for the different frameworks. For example, in the case of Spring Security, around 35% of the issues are resolved between 0 and 20 days after they are raised. If issues found in a security framework are promptly resolved, ART becomes increasingly skewed to the left. For this measure, OACC, Hdiv, OWASP ESAPI, Keyczar, PicketLink produce desirable results. However, OWASP ESAPI and Keyczar had a significant portion of their issues that went unresolved. Therefore, our data can be interpreted as showing that OACC, Hdiv and PicketLink have higher community engagement as they have a low ART and a high percentage of resolved issues.

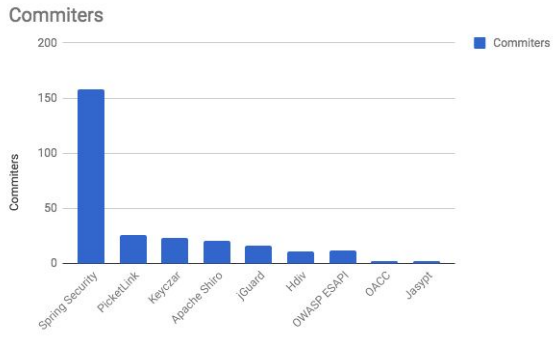

Fig 3: Number of committers

Figure 3 displays the results with respect to the number of committers for the different frameworks. Most of the frameworks, except for Jasypt and Bouncycastle are hosted in GitHub which provides information about the number of committers (called contributors). JGuard provides information about the committers in its homepage. Bouncy Castle, however does not provide clear information about the number of committers and this is why it is not included. JAAS and JCE are part of java so this metric does not apply for them. We can see that Spring Security has the highest number of contributors (158) compared with the other frameworks.

4.3. Evolution

In this paper we use the terms "evolution" and "maintainability" to denote how easy it is for a community of developers to modify a framework to fix bugs (including newly emerging security threats), and to add features (corresponding to new security requirements or variants on existing security requirements). While these terms are not identical, they are strongly related. The degree to which a system can be easily evolved is the degree to which it is easy to find the location of a bug or feature, and independently modify the code responsible for that bug or feature. A system that is not maintainable typically suffers from problems such as high coupling, low cohesion, large monolithic modules, and complex code. All of these characteristics inhibit the evolution of the system, which is why we consider these two terms to be largely interchangeable.

For the purposes of this work we have chosen to examine the architectural complexity of each of our candidate frameworks, as measured by its Decoupling Level (DL)--an architecture-level coupling metric. DL measures how well a system's modules are decoupled from each other and has been shown to strongly correlate with true maintenance costs [7]. This metric can be calculated by the Titan tool suite [11].

Titan takes, as input Design Structure Matrix that contains the dependency relations among project source files. In the examples presented here these dependency relations were generated by a commercial reverse-engineering tool called Understand.[15] Given this input Titan clusters the files and calculates the DL metric based on this clustering. The details of the algorithm and the empirical validation of DL can be found in [7].

Table 5 presents the results of the DL calculations for 10 of the 12 frameworks (we were unable to obtain the source code for JAAS and JCE, and so could not calculate their DL values). With the DL metric, the higher the value the better. The maximum DL value was obtained by JGuard (0.813) while the minimum value was obtained by OWASP ESAPI (0.304). The average DL value over the 11 measured projects is 0.62, which is about the average for all open source projects that we have studied.

But, as with any other metric, DL values by themselves are just numbers and hence difficult to interpret; these numbers must be put into a context. In [7] an analysis of 129 large-scale software projects, covering a broad range of application areas (108 open source and 21 industrial projects) was carried out. 60% of these projects were shown to have DLs

[15] http://www.scitools.com

between 0.46 and 0.75, with 20% having DLs above 0.75 and 20% having DL values below 0.46.

Table 5. Results of the DL Calculations

Project	DL
JGuard	0.813
Bouncy Castle	0.784
Jasypt	0.774
Spring Security	0.737
PicketLink	0.675
Keyczar	0.609
HDIV	0.586
Apache Shiro	0.562
OACC	0.450
OWASP ESAPI	0.304
JAAS	n/a
JCE	n/a

From this data we can conclude that JGuard, Bouncy Castle, and Jasypt are in the top 20% of software projects, in terms of their DL and that OACC and OWASP ESAPI are in the bottom 20%. As mentioned above, DL values for JAAS and JCE could not be obtained.

4.4. Adoption and Popularity

Adoption and popularity are also important criteria for evaluating a security framework. While the respondents to our survey did not rank this as a major criterion, more widespread adoption and popularity, we postulate, implies higher quality, better support, greater likelihood of longevity, and better usability. Hence we included it in our survey as a factor worth measuring. Of course, a more highly adopted and popular framework may be inferior in some other ways, such as having less coverage than newer or less known frameworks. This is, once again, why we have chosen orthogonal measures of framework quality in our evaluation method.

We used Stack Overflow (http://stackoverflow.com) to quantify the security frameworks' adoption and popularity. This website is the most popularly used platform by programmers to discuss technical issues, in the form of Questions and Answers (Q&A). We conducted our searches by using the official names of security frameworks as keywords on the Stack Overflow website, but some name variants were also used. For example, we employed several combinations of keywords--such as 'spring security' and 'spring-security'--to search for postings on the Spring Security framework. Also, we used the advanced search option 'answers:1..' to filter out questions without answers, as we consider these as less relevant. The number of matches to our queries is the number questions posted regarding each security framework. We believe that this is a reasonable approximate measure for the adoption and popularity of a framework.

Figure 4 shows the overall results for the Adoption and Popularity criteria. Among the security frameworks evaluated, Spring Security is by far the most popular with 10453 questions on StackOverflow.

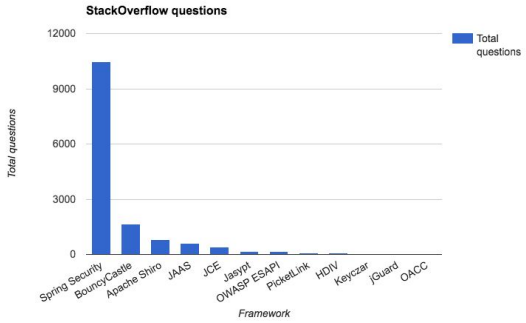

Figure 4: Number of questions on Stack Overflow

5. Discussion

In this section we analyze the results presented in the previous sections and we discuss threats to validity.

5.1. Analysis of results

The data in section 4 shows that it is possible to gather data associated with the criteria presented in section 2 from online sources using a variety of strategies. We believe that the data that we collected could be useful to practicing architects in selecting a security framework, and most of this information can be obtained at low cost, in terms of effort.

Ideally, we would like to gather and process data automatically for all of the criteria, but some of these criteria are difficult to measure by gathering online information or by direct analysis, and some require human intervention. Criteria whose data is difficult to obtain purely from online sources include Ease of Integration, and (total) Cost (of ownership). Functional completeness requires manual data

gathering and processing. While it would be possible to create a tool that performs text analysis of the frameworks' homepages and identify keywords that can be connected to specific tactics (such as "authorization"), this is currently outside the scope of our research, and the feasibility and precision of such an approach is unknown. On the other hand, data for criteria such as Community Engagement, which is an important criterion for architects, can be gathered in a relatively simple and automatable way, based on publicly available information, as we have shown. And other measures, such as maintainability scores, can be fully automated.

5.2. Threats to validity

Some threats to validity of the work presented in this paper include the possibility of having a sampling bias with respect to the frameworks we selected. Security is a quality attribute that is addressed directly by frameworks but other quality attributes such as performance or availability are not directly addressed by specific frameworks; that is, we are not aware of many 'performance' frameworks. We believe, though, that it is possible to find other frameworks for quality attributes such as usability or testability.

The fact that we have only studied security may reflect a confirmation bias, as we may be falsely believing that criterion data can be collected for any framework that can be directly associated with tactics.

One additional threat comes from the fact that the data for the coverage metric had to be collected manually in the sense that it has to be interpreted, as opposed to the data for the other metrics. This may lead to a form of experimenter's bias. While this form of collecting evidence is not optimal, we believe that our approach is justified (and, in fact, unavoidable) in that the coverage metric is essential as it is the only one that focuses on the intent of the frameworks and their users.

Another threat to validity is that, at this point, we have not obtained feedback from practicing architects to understand if they would be willing to make a selection decision based solely on the criteria and the data that we have obtained. We have tried, however, to leverage our extensive experience in working with practitioners to select criteria that we believe would be of use. Still, we might be subject to optimism bias here.

Also, we acknowledge the fact that the size of our survey respondents sample is small. Even though we sent the survey link to several different online groups, we obtained what we consider is a small number of responses. We believe this is due to the fact that our questionnaire requires a fairly specific and hence narrow profile (Java developers who are familiar with, and have participated in the selection of, security frameworks).

Finally, not all of the data can be accurately obtained in a fully automatic way. One of our goals in this work was to automate as much as possible of the measurement process, requiring only easily obtainable, widely used project data. For example, to measure evolution, we used the DL metric, which can be automatically calculated using just the source code. But this is sometimes challenging. For example, it is not possible to differentiate bug fixes from feature requests in some of the frameworks, which is needed to calculate the community engagement metric. And functional completeness requires the input of an expert analyst. Nonetheless, we largely achieved our goal of automated measurement; the functional completeness measure is relatively straightforward and does not need to be repeated for each release of a product.

6. Related work

Our work is unique in the sense that there is a lack of research publications focused on comparing and evaluating software security frameworks. However, there have been attempts to provide general guidance on how to select the best framework irrespective of the application domain. [1] discusses 29 criteria that can be used to evaluate frameworks. Although the list of criteria is useful, the collection of data for many of these criteria (such as "design patterns" or "coupling") is not straightforward. [6] provides an extensive review of the evaluation and selection of software packages, which are complete software systems such as Computer-Aided Software Engineering tools or Enterprise Resource Planning systems. Similar to the previously discussed work, their paper proposes an extensive list of criteria to evaluate software packages. Although software packages are different from frameworks, many of the same criteria can be used when selecting frameworks. Their review, however, does not mention the automated collection of data associated with the packages, which is one important contribution of our work.

The work of [5] presents an empirical study on the selection of open source software. By interviewing 16 developers from different

development companies, the authors arrived at the conclusion that the selection process is often constrained by the situation (for example, company policies) and that the developers use a 'first fit' rather than a 'best fit' approach towards selection. In their opinion, these situations limit the use of more established selection methods. We believe that our approach may help developers avoid the 'first fit' approach by providing them with means to evaluate different alternatives from information that can be gathered, for the most part, in an automated way.

The work of [8] surveys 18 selection approaches for Commercial Off-The-Shelf (COTS) products. From these different approaches, they synthesize a general selection process which considers five steps that are: 1) Define evaluation criteria, 2) Search COTS products, 3) Filter results based on requirements, 4) Evaluate the candidates on the filtered results, 5) Make selection (using decision making techniques). Our work can be of particular use in step 4 of their general selection process.

The topic of evaluating and choosing frameworks has also received considerable attention in the popular press, e.g. [10].

7. Conclusion

In this paper we have performed a study of criteria that are useful to architects in the selection of application frameworks. Our study has allowed us to understand which criteria are important to practitioners and how data associated with some of these criteria can be gathered from online sources.

At this point we have only gathered data for a small number of criteria and our future work includes identifying means to gather data for additional criteria, although this may be challenging for some criteria such as Learnability or Ease of Integration. Future work also includes creating a tool that provides support to architects in the application framework selection process, based on the information that is gathered from online sources.

While the goals of this study are rather narrow--looking at the decisions affecting adoption of security frameworks for Java applications--the methodology that we have applied is not specific to either Java or security. This was simply our initial target. But we believe that the reasoning, criteria, and tools that we have used to collect data in this paper are generic. Thus we can claim that this research represents a first step towards creating scorecards for third-party components, supporting the rapid selection of such components.

8. References

[1] S. Ahamed, A. Pezewski, and A. Pezewski, Towards framework selection criteria and suitability for an application framework," Proceedings International Conference on Information Technology: Coding and Computing (ITCC), 2004, 424-428 Vol.1.

[2] L. Bass, P. Clements, and R. Kazman, Software Architecture in Practice, 3rd edition, Addison-Wesley, 2012.

[3] H. Cervantes, R. Kazman, Designing Software Architectures: A Practical Approach, Addison-Wesley, 2016.

[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design patterns: elements of reusable object-oriented software, Addison-Wesley, 1995.

[5] Ø. Hauge, T. Østerlie, C-F. Sørensen, and M. Gerea, "An Empirical Study on Selection of Open Source Software - Preliminary Results", Proceedings of the 2009 ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development (FLOSS '09), 2009.

[6] A. Jadhav and R. Sonar, "Evaluating and Selecting Software Packages: A review", Journal of Information and Software Technology, 51 555-563, 2009

[7] R. Mo, Y. Cai, R. Kazman, L. Xiao, Q. Feng, "Decoupling Level: A New Metric for Architectural Maintenance Complexity", Proceedings of the International Conference on Software Engineering (ICSE) 2016, (Austin, TX), May 2016.

[8] A. Mohamed, G. Ruhe, and A. Eberlein. COTS Selection: Past, Present, and Future. In ECBS '07 Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, pages 103– 114. IEEE Computer Society, Mar. 2007.

[9] Ryoo, J., Kazman, R., Anand, P., "Architectural Analysis for Security", IEEE Security and Privacy, November/December 2015,13:6, 52-59.

[10] P. Selle, "13 Criteria for Evaluating Web Frameworks", https://www.safaribooksonline.com/blog/2013/10/14/13-criteria-for-evaluating-web-frameworks/, 2013.

[11] L. Xiao, Y. Cai, R. Kazman, "Titan: A Toolset That Connects Software Architecture with Quality Analysis", Proceedings of the 22nd ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE 2014), (Hong Kong), November 2014.