Title: A methodology to support the selection of security frameworks

Authors:
- Jungwoo Ryoo (Member, IEEE), Pennsylvania State University-Altoona, jxr65@psu.edu
- Junsung Cho (non-member), Sungkyunkwan University, js.cho@skku.edu
- Humberto Cervantes (member, IEEE), Universidad Autonoma Metropolitana Iztapalapa, hcm@xanum.uam.mx
- Rick Kazman (Senior Member, IEEE), kazman@hawaii.edu
- Geumhwan Cho (non-member), Sungkyunkwan University, geumhwan@skku.edu
- Jina Kang (non-member), National Security Research Institute, 4558kang@nsr.re.kr
- Hyoungshick Kim (non-member), Sungkyunkwan University, hyoung@skku.edu

Index Terms: Software architecture, Software security, software framework, tactics, design patterns

Abstract:
There are many software frameworks available today, but there is little guidance on how to select the best framework for any given software development project. Without such guidance, users have to conduct their own research and eventually decide on a framework, typically through trial and error, gut feelings, personal experience, or the experience of others. To establish a more principled basis for such decisions, we offer a methodology: an objective set of steps, with associated guidelines, for assessing and comparing the quality and value of frameworks. To demonstrate its use, we apply this methodology by reviewing a set of software security frameworks. We surveyed the most widely used Java-based software security frameworks and compared them based on how well they address key concerns associated with developing secure software: how well they support security functions integrated as part of business logic, as well as other important adoption factors such as their adoption and popularity. In particular we leveraged security tactics to guide our analysis of framework completeness. These tactics help identify and focus scrutiny on essential security features. Our paper benefits potential adopters of security frameworks by providing a methodology and criteria to select an appropriate framework for their context.

Acronyms and Abbreviations:
- AES - Advanced Encryption Standard
- API - Application Programming Interface
- ART - Average Resolution Time
- COTS - Commercial Off-The-Shelf
- DL - Decoupling Level
- JAAS - Java Authentication and Authorization Service
- JCE - Java Cryptography Extension
- JEE - Java Enterprise Edition
- JVM - Java Virtual Machine
- OACC - Object Access Control Framework
- OWASP ESAPI - Enterprise Security API

1. Introduction

The activity of designing a software architecture involves making design decisions to satisfy architectural *drivers* [Bass 2012]. Drivers are requirements which have a direct influence on the success of a software system, and hence on its architecture. In this paper we are interested in examining the satisfaction of one category of driver--quality attributes--and more specifically we are interested in the quality attribute of security.

When designing to address security concerns, it makes sense for an architect to attempt to reuse existing solutions. These solutions include design concepts such as design patterns, tactics or externally developed components such as frameworks [Cervantes2016b]. Design patterns are conceptual solutions to recurring design problems and security design patterns focus on solutions for security [Fernandez2013]. Tactics are proven design strategies that influence the control of a quality attribute response [Bass 2012]. Frameworks are reusable software elements that provide generic functionality addressing a recurring domain and quality attribute concern, across a broad range of applications [Cervantes2013]. For instance, if an architect has a security requirement for resisting attacks, he can start by identifying and selecting the relevant tactics (see Figure 1). The chosen tactics may be design strategies such as: Authorize Actors and Authenticate Actors. Next, the architect must decide on how to implement these tactics. At this point there are several options:

a) Create an entirely *new* ad-hoc solution from scratch: while this new solution might be better than existing solutions, it is often less-than-optimal from the perspectives of cost, schedule, and (frequently) quality.
b) Use a security pattern: This may be more efficient and informed than a) but it still requires the pattern to be refined into code. Furthermore, multiple patterns are typically involved in the satisfaction of a security concern (for example, there is no point doing authentication if you do not also do authorization), which makes this task nontrivial.
c) Use an application framework: this can be the most efficient approach as the architect reuses a well-tested solution that solves the security problem. A framework is reusable, tested code that already implements a number of patterns and tactics.

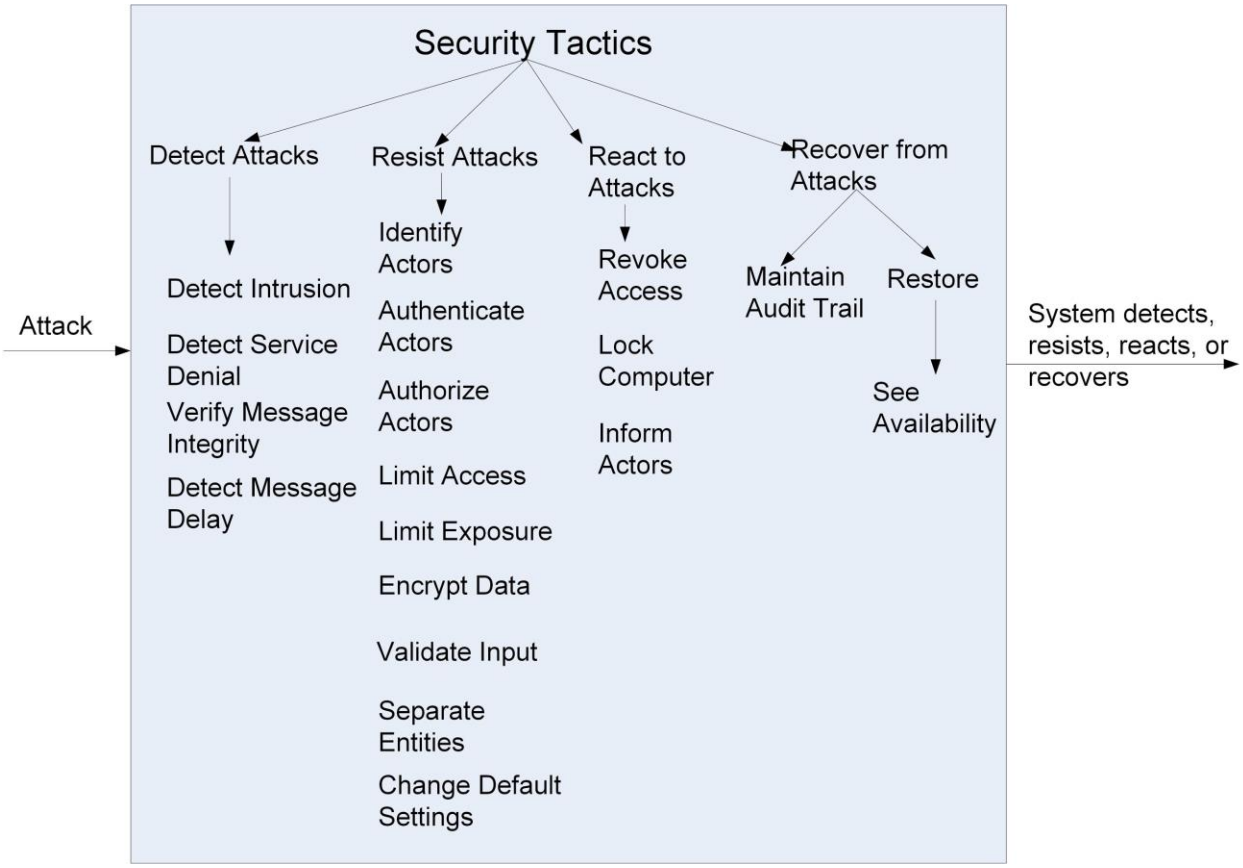


Figure 1: Security Tactics Hierarchy

In [Cervantes2016a], the analysis of several web-based applications demonstrated that the use of frameworks is an appropriate and cost-effective approach to addressing security. However, a significant challenge associated with this approach resides in the selection of an appropriate framework for a given set of requirements. There are many frameworks--both commercial and open-source--that address security, but there is little guidance available to architects on making decisions about which framework to adopt. This is a critical decision. Choosing a framework, even if it is “free” and open source, involves a commitment of substantial resources: to master the learning curve, to integrate the chosen framework, and to test the integrated system. Thus, the decision of a framework should not be taken lightly and should not be made based on gut feeling or fashion.

In this paper we propose a more reasoned, disciplined approach to this critical decision. We first establish a set of criteria to help in selecting appropriate frameworks to address security requirements, and then perform a rigorous analysis of different security frameworks using these criteria.

The structure of the paper is as follows. Section 2 discusses concepts associated with frameworks in general and presents the particularities of security frameworks. Section 3 presents the goals and the scope of the research. Section 4 presents the selected criteria and the procedures followed to collect data. Section 5 presents the results from the data collection

and section 6 discusses the methodology for framework selection. Section 7 presents a discussion on threats to validity. This is followed by a presentation of related work in section 8 and, finally, section 9 concludes the paper and discusses future work.

2. Software Security Frameworks

In this section we discuss about application frameworks in general and about security frameworks.

2.1 Application frameworks

An application framework (or simply *framework*) is a collection of reusable software elements, constructed out of patterns and tactics, which provide generic functionality addressing recurring domain and quality attribute concerns across a broad range of applications. Frameworks, when carefully chosen and properly implemented, help increase the productivity of programmers. They achieve this by allowing programmers to focus on business logic and end-user value, rather than on addressing recurring problems which have already been solved [Cervantes2013]. Examples of these recurring problems include performing object-oriented to relational mapping, creating user interfaces or authenticating users. As opposed to self-contained products, framework functions are generally invoked from the application code, or they are "injected" into the application code using some type of aspect-oriented approach. Frameworks usually require configuration, typically through XML files or through other approaches such as annotations in Java or following certain conventions in Ruby on Rails. An example framework is Hibernate (http://hibernate.org/) which is used to perform object-oriented domain model to relational database mapping in Java, mainly by adding anotations to the Java code.

There are two major categories of frameworks: "full stack" frameworks, such as Spring (https://projects.spring.io/spring-framework/) or Java Enterprise Edition (JEE: http://www.oracle.com/technetwork/java/javaee/overview/index.html), which are usually associated with reference architectures and address general concerns across the different layers and elements of the reference architecture. On the other hand non-full stack frameworks, such as Java Server Faces (http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html), are narrower in scope, addressing specific functional or quality attribute concerns. Furthermore, frameworks can be "standalone" or "non-standalone". The former can be deployed independently, and have to be included as part of the application, for example in a jar file, while the latter are part of the API and do not need to be explicitly included as part of the application.

2.2 Security Frameworks

There are many software frameworks that address security concerns. For example, full-stack frameworks such as JEE include APIs that address authorization and authentication aspects of security (JAAS: http://www.oracle.com/technetwork/java/javase/jaas/index.html). Other frameworks are oriented towards addressing specific concerns, such as the creation of user interfaces, but also address aspects of security. For instance, the ZK framework (https://www.zkoss.org/) is oriented towards the creation of user interfaces for web applications but also addresses security attacks that commonly occur via the user interface, such as SQL

injection attacks [Potix2015]. Finally, there are also stand-alone frameworks that are dedicated specifically to addressing security requirements. The Spring Security framework (http://projects.spring.io/spring-security/) is a good example of this category.

Software security frameworks are typically language-specific. Depending on the popularity of the language, there will be different numbers of frameworks and, of course, different numbers of frameworks specifically addressing security. For instance, according to our research, there are considerably more Java frameworks supporting security than there are for other programming languages. Finally, we can categorize software security frameworks according to whether they are open-source or proprietary.

2.3 Selecting frameworks

The selection of externally developed components such as frameworks is an important decision that is made as part of the architecture design process. The selection of frameworks can be a challenging task because of their extensive number. There are a few criteria that are considered by architects when selecting externally developed components:

- Problem that it addresses - is it something specific, such as a framework for object oriented to relational mapping or something more generic, such as a platform?
- Cost - what is the cost of the license and, if it is free, what is the cost of support and education?
- Type of license - does it have a licence that is compatible with the project goals?
- Support - Is it well supported? Is there extensive documentation about the technology? Is there an extensive user or developer community that you can turn to for advice?
- Learning curve - how hard is it to learn this technology? Are there courses available?
- Maturity - Is it a technology that has just appeared on the market, which may be exciting but still relatively unstable or unsupported?
- Popularity - is it a relatively widespread technology? Are there positive testimonials or adoption by mature organizations? Will it be easy to hire people who have deep knowledge of it? Is there an active developer community or user group?
- Compatibility and ease of integration - is it compatible with other technologies used in the project? Can it be integrated easily in the project?
- Support for critical quality attributes - does it limit attributes such as performance? is it secure and robust?
- Size - will the use of the technology impact negatively on the size of the application under development?

Unfortunately, the answers to these questions are not always easy to find and the selection of a particular technology is frequently performed using ‘first fit’, rather than a ‘best fit’, approach [Hauge2009]. This means that developers will search alternatives based on their experience and from internet searches. Once they identify an option that can solve their problem, they download and test the candidate and if it satisfies their needs, they stop searching for more options. We believe that one of the problems of selection is that in general, it takes a considerable effort to evaluate multiple alternatives, and this promotes the use of a ‘first fit’ approach. Providing mechanisms that facilitate evaluating multiple alternatives with less effort is an essential aspect of this research.

3. Research Goals and Scope

In this section we discuss the goals and the scope of the research.

3.1 Research goals

In this study we are pursuing the following research goals:

1. **RG1:** Develop a practical set of criteria to facilitate the selection of security frameworks
2. **RG2:** Collect data to support the analysis of the criteria identified in RG1
3. **RG3**: Propose a methodology that allows architects to use the criteria in RG1 to select the best frameworks for their security needs

Note that in RG1 we specify, as part of the research goal, that the set of criteria are "practical". By practical we refer to the fact that the data associated with these criteria can be collected in a relatively easy way. It should be noted that any data associated with the criteria that we collect at the time of writing this paper is time-sensitive and may, in fact, be incorrect by the time the paper is published. Over time every one of the frameworks that we examine will evolve, some will disappear, and new ones will emerge. So the ultimate goal and output of this paper is not to identify specific frameworks which are "better" or "worse", but rather to illustrate how one might evaluate frameworks in general, to support their selection.

3.2 Scope

For the purpose of addressing the research goals 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. 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. This constraint also allows us to provide a deeper analysis of each selected framework. Table 1 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 (6 frameworks), cryptography (4 frameworks), or protection of web applications against attacks (2 frameworks). The descriptions in table 1 come from the frameworks homepages.

ID	Name and version	Main focus	Description
1	Spring Security[1] (v 4.1.0)	Authorization and authentication	Spring Security is a powerful and highly customizable authentication and access-control framework. It is used in conjunction with the Spring framework and is the de-facto standard for securing Spring-based applications.

[1] http://projects.spring.io/spring-security/

2	Bouncycastle[2] (v 1.54)	Cryptography	Java cryptography APIs.
3	JAAS[3] (Java SE 8)	Authorization and authentication	The Java Authentication and Authorization Service is part of the Java security APIs. It is focused on authorization and authentication.
4	JCE[4] (Java SE 8)	Cryptography	The Java Cryptography Extension is part of the Java security APIs. It is focused on cryptography.
5	Apache Shiro[5] (v 1.2.4)	Authorization and authentication	Powerful and easy-to-use Java security framework that performs authentication, authorization, cryptography, and session management.
6	Jasypt[6] (v 1.9.2)	Cryptography	Java library which allows the developer to add basic encryption capabilities to his/her projects with minimum effort, and without the need of having deep knowledge on how cryptography works.
7	HDIV[7] (v 2.1.11)	Protection of web applications against attacks	HDIV is an 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.
8	OWASP ESAPI[8] (v 2.1.0)	Protection of web applications against attacks	This is the Java EE language version of OWASP ESAPI (Enterprise Security API).
9	Keyczar[9] (v 0.71)	Cryptography	Keyczar is an open source cryptographic toolkit designed to make it easier and safer for developers to use cryptography in their applications. Keyczar supports authentication and encryption with both symmetric and asymmetric keys.
10	OACC[10] (v 2.0.0)	Authorization and authentication	OACC (pronounced Oak) is an advanced Java Application Security Framework. OACC provides a high performance API that provides permission-based authorization services for Java applications. In a nutshell, OACC allows your application to enforce security by answering the question: Is entity 'A' allowed to perform action 'p' on entity 'B'?

[2] http://www.bouncycastle.org/
[3] http://www.oracle.com/technetwork/java/javase/jaas/index.html
[4] http://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html
[5] http://shiro.apache.org/
[6] http://www.jasypt.org/
[7] http://www.hdiv.org/
[8] https://github.com/ESAPI/esapi-java-legacy
[9] https://github.com/google/keyczar
[10] http://oaccframework.org/

11	jGuard[11] (v 1.0.4)	Authorization and authentication	jGuard is a library that provides easy security (authentication and authorization) for Java web applications. It is based on the stable and mature JAAS framework, which is part of the Java Standard Edition API.
12	PicketLink[12] (v 2.7.1)	Authorization and authentication	PicketLink provides an alternative to JEE Security, providing a rich, powerful and flexible API to secure your applications.

Table 1. Security frameworks considered in the scope of this paper

4. Framework Evaluation Criteria

This section addresses RG1: Develop a practical set of criteria to facilitate the selection of security frameworks.

4.1 Selection of criteria

As discussed in section 2.3, there are many ways to characterize and evaluate any externally developed component, including frameworks. Choosing criteria is a non-trivial problem. A criterion such as “framework usability”, while important for long-term success, is extremely time-consuming to assess and, unless your organization has the resources to do a large-scale controlled study, will likely not be objective. It is not uncommon to observe comments in discussion forums where people argue that a particular framework is “easier to use” than some other one. Usability is relative to the complexity of the Application Program Interface (API), but this complexity can be mitigated by the knowledge of the programmer. No matter how comprehensive or popular a security framework is, it gets rarely used if its usability is poor. While it would be desirable to measure usability, in practice this criterion is difficult to evaluate. There have been some attempts to measure usability in an automated way [Scheller2015] and [Piccioni2013] but so far these studies have had a limited scope.

Similarly, “quality of documentation” is difficult to objectively assess without a large-scale controlled study. While both usability and quality of documentation are doubtless important to the success of a framework, we suggest that an organization that needs to choose among a set of frameworks use our proposed methodology to winnow the set down to a small number of candidates and then subjectively assess other criteria, like usability, to make a final choice. Other criteria might also be relevant in the final choice such as cost, familiarity of the development team with a particular framework, the framework’s license, effect on other quality attributes (such as performance or reliability), effect on the size of the resulting executable, and so forth.

For this study we have endeavored to identify criteria that satisfy the following goals:

[11] http://jguard.xwiki.com/xwiki/bin/view/Main/WebHome
[12] http://picketlink.org/appsecurity/

1. The criterion is objective. A criterion such as ‘framework usability’ may be difficult to measure, as this is largely subjective. It might even be difficult to get broad consensus on the meaning of such a criterion.
2. Data to measure the criterion can be obtained in a relatively simple, economical way. Our broader goal for this research is to establish a methodology that allows frameworks--any frameworks--to be compared to one another. New frameworks appear constantly, so it should be easy to obtain information that allows comparisons to be made and to update existing values of the criteria.
3. The criterion is orthogonal to other criteria already chosen. All things being equal, we would like each of our criteria to measure a different dimension of framework goodness.

Table 2 describes the initial set of criteria that we identified, their meaning, and the rationale for their selection.

Criterion	**Meaning**	**Rationale**
Completeness	How many areas of security concerns are addressed by the framework?	An architect who needs to address many security requirements will probably favor a framework with high completeness, but if only one aspect of security needs to be addressed (e.g. encryption), it may be better to use a more specialized framework. It is also possible to use multiple frameworks, but this could have negative consequences, such as a higher learning curve, managing inter-dependencies, and increases in the codebase size.
Adoption and Popularity	How widespread is the use of the framework?	A framework that has broader user acceptance should have better support, better longevity, and higher quality. Such frameworks will probably be favored over other, less broadly accepted ones.
Maintainability	Is the framework internally well designed and easy for the developers to maintain?	A framework that not well designed will, over time, be harder and harder to maintain, extend, and debug. Selecting a well designed framework is preferable to avoid current and future quality issues.
Community Engagement	Is the framework maintained properly? How active is the community that develops it?	A community that is well engaged will strive to resolve issues that are discovered and will also resolve this issues in a relatively short time period. Furthermore, an active community has a considerable number of contributors.

		Frameworks whose community is not engage will eventually lose their popularity, resulting in low adoption. Also, security frameworks whose issues are not solved in an opportune time window may be vulnerable to security attacks, defeating their very purpose.

Table 2. Selected criteria

We believe that the criteria presented in Table 2 meet the three goals described above. In the following subsections we discuss the metrics associated with each of these criteria, along with the procedures used to collect the data.

4.2 Completeness

We measure 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. However, more may not necessarily be better: it may be infeasible or unwise for a single framework to attempt to address every possible security feature. Specialization could, in fact, make a framework more effective or practical.

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 [Bass2012]. Tactics have been used to guide both design and analysis [Cervantes2016b] [Ryoo2015]. The way that we employ tactics here is 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 completeness of the framework. In this way we will be able to rank the frameworks according to the degree of each framework's coverage of security tactics. But this must be interpreted with care: *horizontal coverage* (or breadth, measured in the number of security tactics addressed by a particular framework) is not everything. The depth or quality of coverage--what we term *vertical coverage* (how well, or in how many different ways, each security feature is covered)--is an equally important criterion.

4.2.1 Security Tactics and Horizontal Coverage

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 coding or by employing a software component such as a framework.

By understanding which specific tactics are addressed by a security framework, we measure its horizontal 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 horizontal coverage of the framework is quite limited as it only covers a single tactic.

To measure horizontal 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.

To ensure the consistency across our data collection, the information was captured using a template, as shown in 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 APIs used to implement the tactic, and additional justifications for how a decision on a framework's support for the tactic was made. In Table 3 we provide an example of how this template was filled out for Spring Security, for the "Authenticate Actors" tactic.

#	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. You should preferably use the framework's official documentation to fill this section.	If you answered Yes/Not Sure, please list the packages in the framework API, which are associated with the tactic. You must use the framework's 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	org.springframework.security.authentication org.springframework.security.authentication.dao org.springframework.security.authentication.encoding org.springframework.security.authentication.event org.springframework.security.authentication.jaas ...

				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	

Table 3: Template used to capture horizontal coverage information

4.2.2 Security Tactic Instances and Vertical Coverage

Irrespective of the level of horizontal coverage, we are also interested in the vertical coverage of each tactic. A framework may specialize in just a handful of tactics, but provide particularly deep coverage of those tactics. An adopter of security frameworks, knowing this, can then make a reasoned decision about whether to choose several narrow but deep frameworks (i.e. ones that provide limited horizontal coverage but extensive vertical coverage), and be faced with the additional challenge of learning and integrating these frameworks. Or the adopter might prefer to choose a single framework (like Spring Security or OWASP ESAPI) that gives broad horizontal coverage but less vertical coverage in many areas.

An important question to ask here is how to measure this dimension of coverage. We propose measuring this using the concept of (security) tactic *instances*, which are concrete instantiations of the leaf nodes of the security tactics hierarchy in Figure 1. For example, the '*Encrypt Data*' tactic might be instantiated as '*Encrypt Data Using Advanced Encryption Standard (AES)*' and '*Encrypt Data Using RSA*.' The number of tactic instances of a security framework tactic is thus a measure of vertical coverage. Note that we are simply counting here; we are not measuring the quality of these instances. Quality is measured by other (orthogonal) criteria, such as maintainability and community engagement.

To obtain vertical coverage data we explored the tactic instances in each framework by examining its documented APIs. For instance, Bouncy Castle offers APIs such as org.bouncycastle.jcajce.provider.asymmetric.rsa to implement an asymmetric encryption algorithm called RSA. The API is then mapped to a corresponding security tactic. It should be noted that it is currently necessary to rely on a human expert to perform this mapping, since such a decision requires significant domain knowledge.

4.3 Adoption and Popularity

Adoption and popularity are also important criteria for evaluating a security framework. More widespread adoption and popularity, we postulate, implies higher quality, better support, greater likelihood of longevity, and better usability. 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.

4.4 Maintainability

In this paper we use the term "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).

The maintainability of a software system depends on the inherent maintainability of the code (which can be measured by code complexity metrics) and the inherent complexity of its architecture (which affects how easy it is for developers to work independently with confidence that their changes will have little effect on each other). 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). DL measures how well a system's modules are decoupled from each other and has been shown to strongly correlate with true maintenance costs [Mo2016]. This metric is calculated by the Titan tool [Xiao2014].

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[13]. 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 [Mo2016].

4.5 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 both programming bugs to be fixed and feature enhancement requests. We consider that a high resolution ratio indicates that the community that develops the framework actively 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 solved. This measure is calculated by $Tr(i)$- $Tp(i)$

[13] https://scitools.com/

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 (i.e. committers). A high number of contributors also indicates that there is a vigorous community continuing the development of the framework.

It should be noted that 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 are using 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.

5. Data collection

In this section we address research goal RG2, namely, the collection of data to support the analysis of the criteria. We present the results that we collected over our 4 criteria and our 12 chosen security frameworks.

5.1 Completeness

We first present the data collected for the completeness criteria which is decomposed into horizontal and vertical coverage.

5.1.1 Horizontal Coverage

Table 4 summarizes the horizontal coverage of all the security frameworks we considered in this study (Y's mean the tactic is covered by the framework). Note that in the table we omitted the tactics not covered by *any* of the frameworks we reviewed. 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 1).

	Resist Attacks						Detect Attacks		Recover from Attacks
	Identify Actors	Authenti-Cate Actors	Autho-rize Actors	Encrypt Data	Limit Access	Validate Input	Verify Message Integrity	Detect Intrusion	Maintain Audit Trail
JGuard	Y	Y	Y		Y				
OACC	Y	Y	Y		Y				
PicketLink	Y	Y	Y		Y				
JAAS	Y	Y	Y		Y				
Jasypt				Y			Y		
Keyczar				Y			Y		
JCE				Y			Y		
Bouncy Castle				Y			Y		
OWASP ESAPI	Y	Y	Y	Y	Y	Y	Y	Y	Y
Spring Security	Y	Y	Y	Y	Y				
Apache Shiro	Y	Y	Y	Y	Y				
HDIV						Y		Y	Y

Table 4. Horizontal Coverage for the different frameworks.

5.1.2 Vertical Coverage

We now present the framework evaluation data associated with vertical coverage according to the three different framework groups discussed in the previous section. Note that although vertical coverage information was obtained by analyzing the frameworks' APIs, we do not list the names of the individual APIs in Tables, 5, 6, and 7 for brevity reasons. For the first group, which consists of frameworks whose primary focus is on cryptography, our analysis results are shown in Table 5. Note that a Y in a table cell indicates that the framework contains APIs addressing the specific tactic instance. As this table shows, Bouncy Castle has the highest vertical coverage in this group.

Tactic Instance	Bouncy Castle	JCE	Jasypt	Keyczar	Associated Tactic
ASN.1 Support	Y				Encrypt Data

Certificate Packages	Y	Y			Encrypt Data
Cipher	Y	Y	Y	Y	Encrypt Data
CMS	Y				Encrypt Data, Verify Message Integrity
Key Management	Y	Y			Encrypt Data
Message Digest	Y	Y	Y		Verify Message Integrity
MAC	Y	Y		Y	Verify Message Integrity
OpenSSL and PEM Support Packages	Y				Encrypt Data
Post-Quantum Crypto	Y				Encrypt Data
Signers	Y	Y		Y	Verify Message Integrity
TLS	Y	Y			Encrypt Data
TSP Packages	Y				Encrypt Data

Table 5: Vertical Coverage for the framework group focused on cryptography

For the second group of frameworks, which is composed of those focusing on authorization and authentication, the analysis results are shown in Table 6. For this group, JGuard and JAAS have the highest vertical coverage.

Tactic Instance	JGuard	OACC	PicketLink	JAAS	Associated Tactic
Access control model	Y	Y	Y	Y	Limit Access
Credential	Y	Y	Y	Y	Authenticate Actors
Login	Y	Y	Y	Y	Authenticate Actors
Policy	Y			Y	Authorize Actors
Auth Permission	Y	Y	Y	Y	Authorize Actors
Private Credential Permission	Y			Y	Authorize Actors

Table 6: Vertical Coverage for the frameworks focused on Authentication and Authorization

For the third group of frameworks, those which provide a comprehensive set of security features, the analysis results are shown in Table 7. In this group, OWASP ESAPI has the highest vertical coverage.

Tactic Instance	OWASP ESAPI	Spring Security	Apache Shiro	HDIV	Associated Tactic
Access control	Y	Y	Y		Limit Access
Credential		Y	Y		Authorize Actors
Login	Y	Y			Authenticate Actors
Policy	Y				Authorize Actors
Auth Permission		Y	Y		Authorize Actors
Input validation	Y			Y	Validate Input
Output encoding/escaping	Y				Validate Input
Cipher	Y	Y	Y	Y	Encrypt Data, Verify Message Integrity
Key Management	Y	Y			Encrypt Data
Message Digest		Y	Y		Verify Message Integrity
Error handling and logging	Y			Y	Maintain Audit Trail
Web Application Firewall	Y				Detect Intrusion
OpenID		Y			Identify Actors
LDAP		Y			Identify Actors

Table 7: Vertical coverage for the frameworks with comprehensive support

5.2 Adoption and Popularity

Figure 2 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.

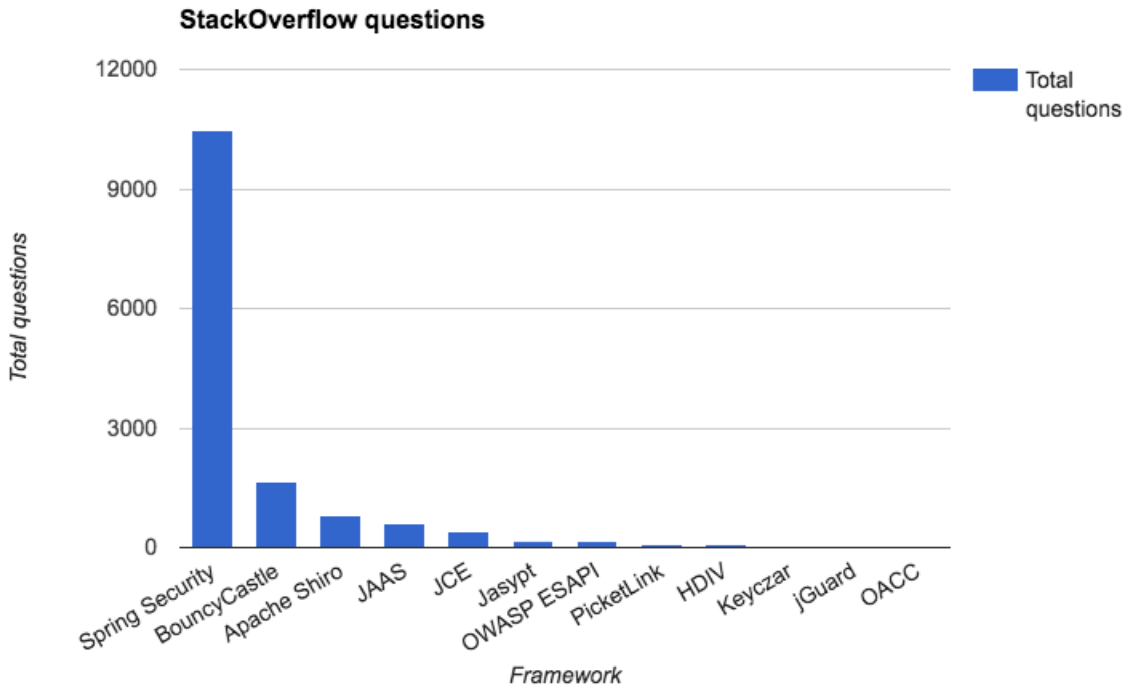


Figure 2: The number of questions for the different security frameworks on Stack Overflow (captured on June 23d, 2016)

We also analyzed the evolution of the number of questions over time and the results are shown in table 8. Excluding the results from 2016 (for which we have only incomplete data), it seems that Spring Security is the framework with the strongest growth trend. It is interesting, and somewhat surprising, to observe the difference in the total number of questions between Spring Security and all the other frameworks.

Framework	...2009	2010	2011	2012	2013	2014	2015	2016…	Total questions	Search term used
Spring Security	140	461	1055	1440	1755	2196	2386	1020	10453	“Spring Security”, "springsecurity"
Bouncy Castle	44	114	227	295	299	284	294	113	1670	“Bouncy Castle”, "BouncyCastle"
Apache Shiro	5	30	71	125	189	197	120	58	795	“Shiro”, "ApacheShiro"
JAAS	42	53	90	103	98	113	78	29	606	“JAAS”
JCE	16	33	53	66	75	80	68	30	421	“JCE”
Jasypt	1	12	11	25	41	36	37	17	180	“Jasypt”
OWASP ESAPI	2	3	19	35	23	47	32	15	176	“ESAPI”
PicketLink	0	1	2	3	10	27	26	4	73	“PicketLink”, "Picket Link"
HDIV	0	0	2	11	4	11	37	3	68	“HDIV”

Keyczar	5	2	4	3	6	3	1	0	24	“Keyczar”
jGuard	1	0	0	0	1	0	0	0	2	“jGuard”
OACC	0	0	0	0	0	0	0	0	0	“OACC”

Table 8.Changes in the number of questions about security frameworks on Stack Overflow

5.3 Maintainability

Table 9 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.

Project	Date	DL
JGuard	5/24/2016	0.813
Bouncy Castle	4/17/2016	0.784
Jasypt	5/24/2016	0.774
Spring Security	4/25/2016	0.737
PicketLink	4/18/2016	0.675
Keyczar	5/5/2016	0.609
HDIV	4/25/2016	0.586
Apache Shiro	4/25/2016	0.562
OACC	5/24/2016	0.450
OWASP ESAPI	4/25/2016	0.304
JAAS	n/a	n/a
JCE	n/a	n/a

Table 8. Results of the DL Calculations (note: higher DL is better), snapshots from May 2016

But DL values by themselves are just numbers and hence difficult to interpret; these numbers must be put into a context. In [Mo2016] 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 between 0.46 and 0.75, with 20% having DLs above 0.75 and 20% having DL values below 0.46.

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%. Data for JAAS and JCE could not be obtained.

5.4 Community Engagement

Figure 3 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).

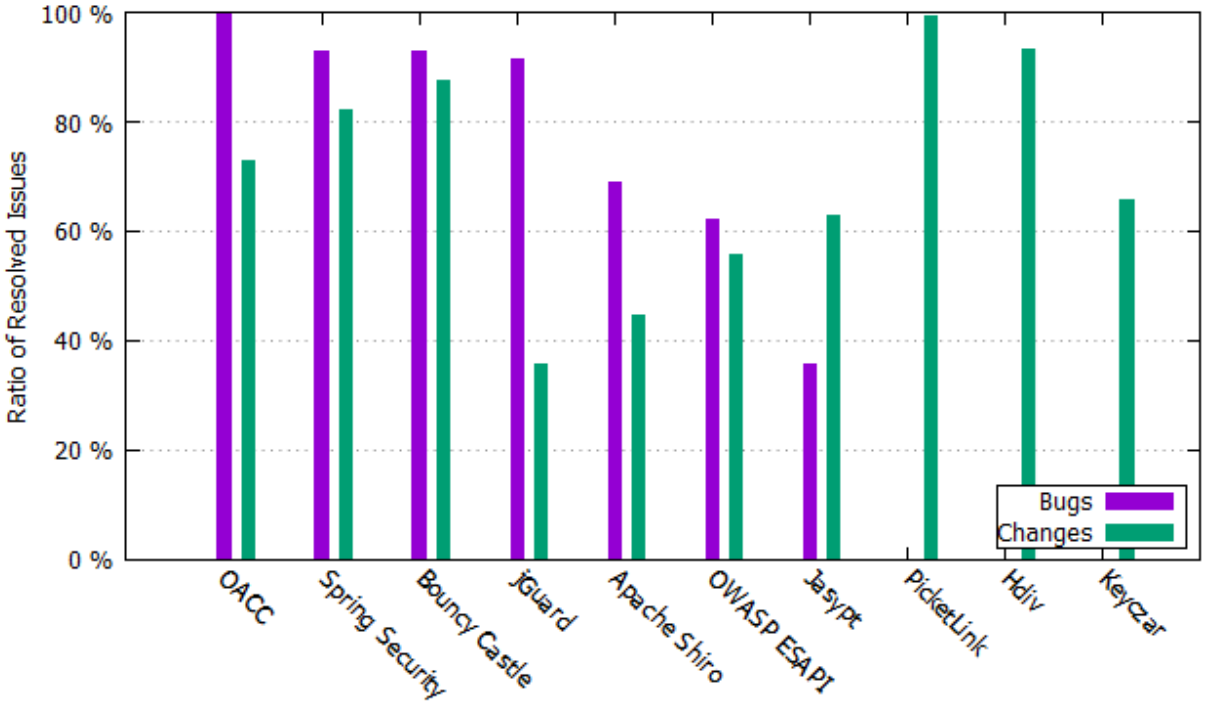
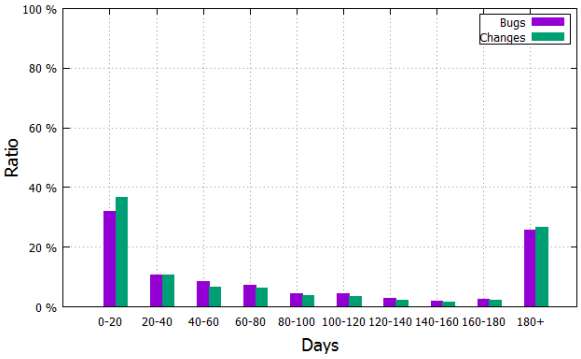
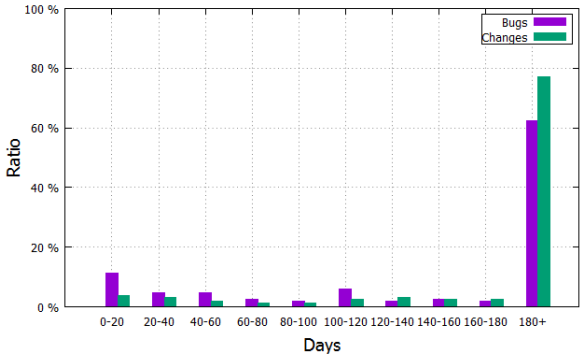


Fig 3: Ratio of resolved issues for security frameworks

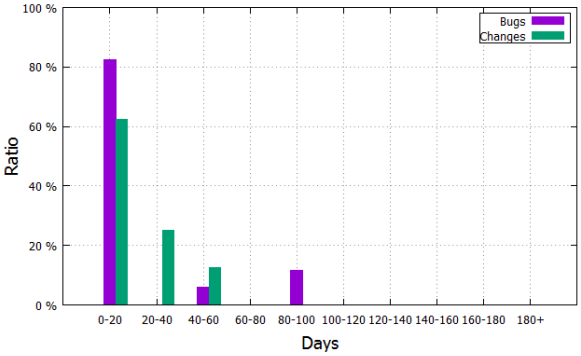
Figure 4 shows the distribution of the percentage of issues resolved across time periods (in days) for the different frameworks. For example, in the case of Spring Security, around 35% of the issues are resolved between 0 and 20 days after they are raised. If issues found in a security framework are promptly resolved, ART becomes increasingly skewed to the left. For this measure, OACC, Hdiv, OWASP ESAPI, Keyczar, PicketLink produce desirable results. However, as shown in Figure 3, OWASP ESAPI and Keyczar demonstrated that a significant portion of their issues are still unresolved. Therefore, our data can be interpreted as showing that OACC, Hdiv and PicketLink have lower community engagement as they have a low ART and a high percentage of resolved issues.



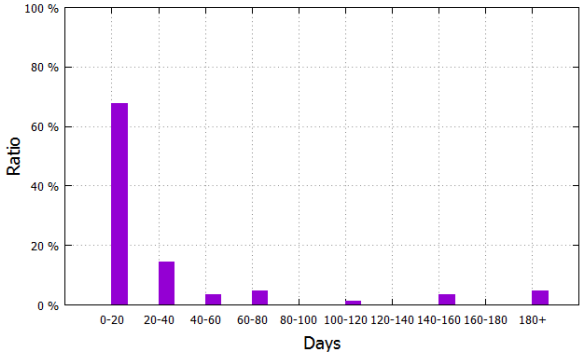
(a) Spring Security



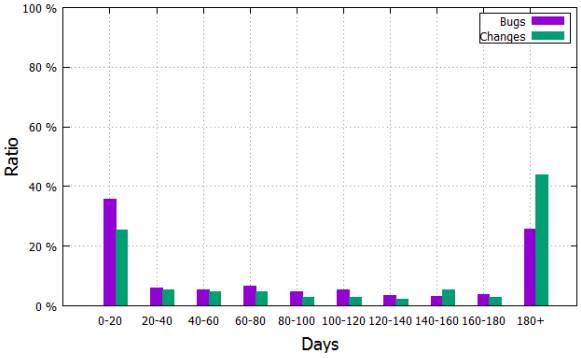
(b) Apache Shiro



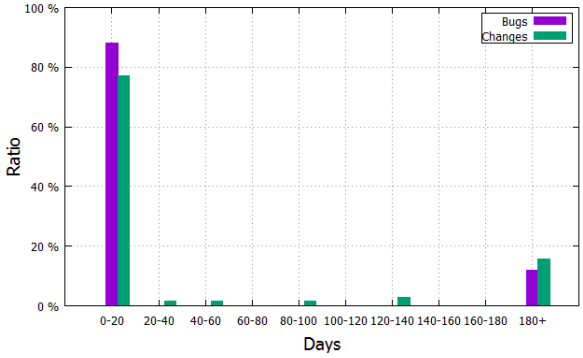
(c) OACC



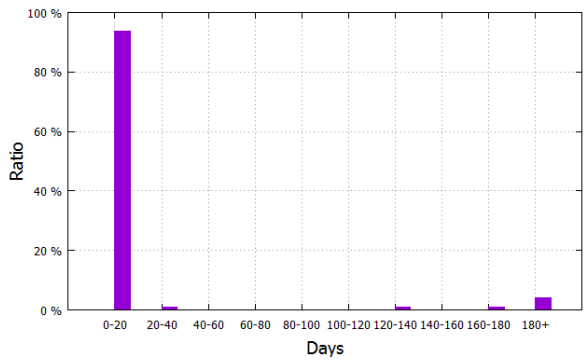
(d) Hdiv



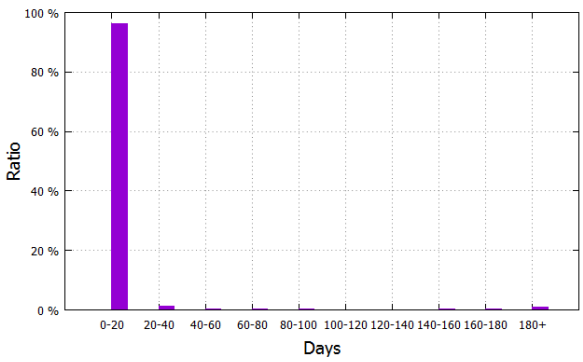
(e) Bouncy Castle



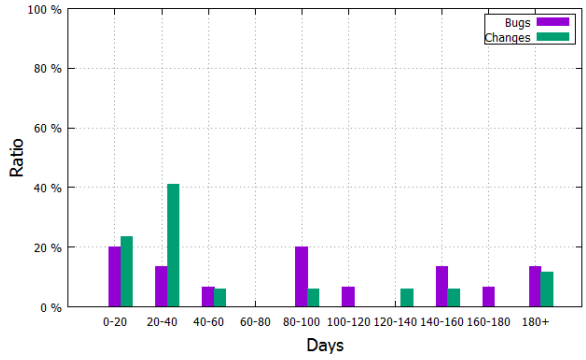
(f) OWASP ESAPI



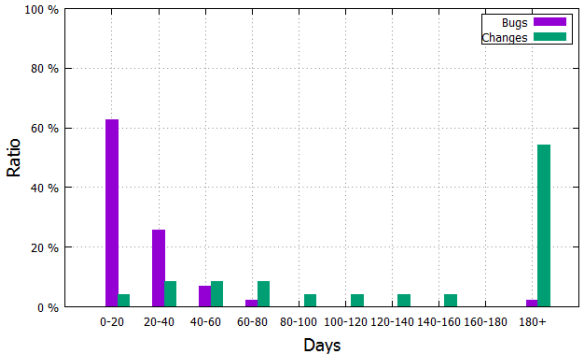
(g) Keyczar



(h) PicketLink



(i) Jasypt



(j) jGuard

Fig 4: Distribution of issue resolution time for the security frameworks

Figure 5 displays the results with respect to the number of contributors (i.e. commiters) for the different frameworks. In the case of Spring Security, PicketLink, Keyczar, Hdiv, OWASP ESAPI, and OACC, we can readily get the contributor information from GitHub. Also, jGuard and Jasypt show their contributors on their sourceforge.net website. Apache Shiro and Bouncy Castle list their contributors on their own project websites. In the case of Bouncy Castle we did not find the number of committers, as opposed to the other frameworks. However, as we mentioned, the number comes from the framework homepage where they list every single person that has contributed. Excluding BouncyCastle, we can see that Spring Security has the highest number of contributors (119) compared with the other frameworks.

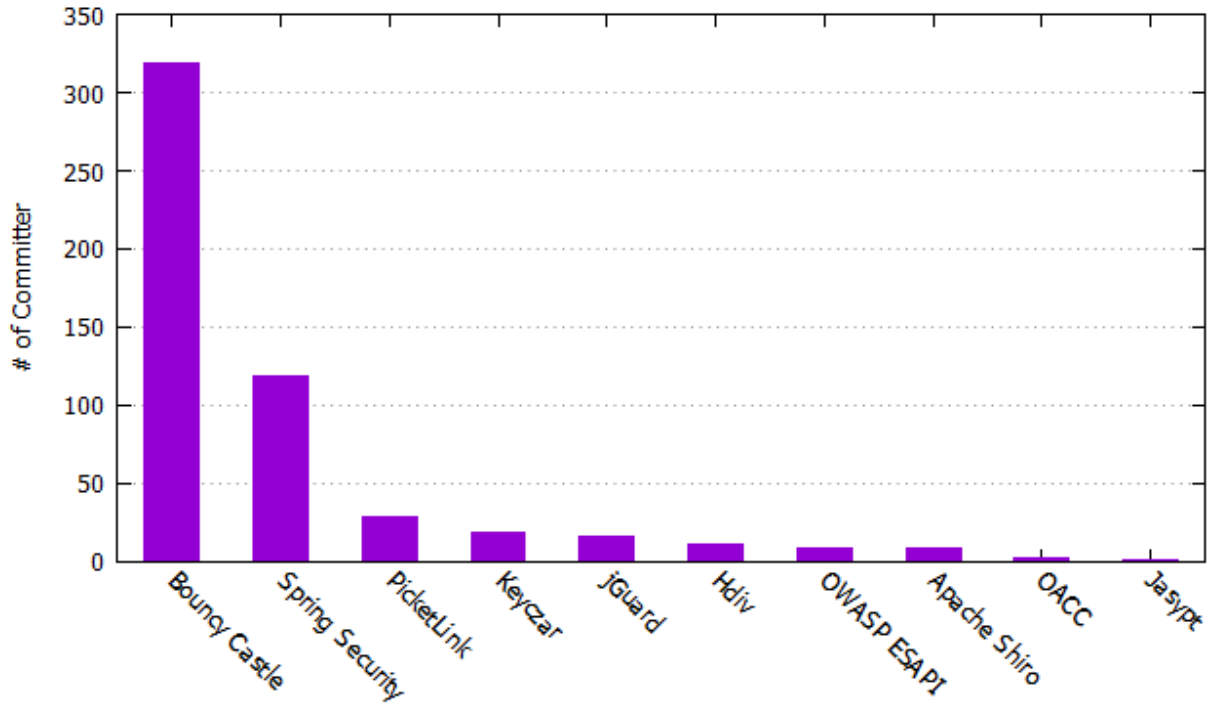


Fig 5: Number of Contributors

6. A Methodology for Framework Selection

In this section we address research goal RG3, that is proposing a methodology that allows architects to use the criteria in RG1 to choose the best framework for their security needs. We also discuss how framework evaluation--while important and valuable--is complex and typically given inadequate attention in practice.

In the end, our methodology for framework selection is the major contribution of this paper. The specific frameworks that we chose to evaluate, the specific evaluation criteria, and the mapping of the criteria onto the chosen frameworks serve as an extended example of this methodology and an existence proof that the method is usable.

The methodology itself is simple, generic, and repeatable. Although we have focused on security frameworks, this methodology could in principle be used to evaluate any set of frameworks that address a common set of concerns.

Our methodology consists of four major phases:

0. Identify candidate frameworks
1. Develop a practical set of criteria for framework selection
2. Collect data to support the analysis and evaluation of the identified criteria
3. Select a framework based on the evaluated criteria

While Phase 0 is obviously important, it is difficult to automate this phase, or even to provide clear guidance on its operationalization. The evaluation team must simply satisfy themselves that they have done an exhaustive search for relevant candidate frameworks.

Phase 1 is the most subtle and most important, which is why we have devoted a great deal of space in this paper to it. The chosen criteria must reliably predict the likelihood of success of the framework for the organization and its requirements, both in the near-term and the longer-term. Furthermore, these criteria must be objective and practical to collect.

Data collection, as exemplified in section 5, needs not be overly time-consuming. But the team must keep in mind that the evaluation of completeness, as discussed in section 4.2, will require the effort of a human analyst. The tactics categorization can guide this process, but there is still human judgement involved. For this reason, we recommend that it be done by several independent analysts who can then compare their results and arrive at a consensus opinion.

Finally, once data for the different criteria has been obtained, the architect can select the framework that better suits his or her needs. This final step requires a value judgement on the part of the evaluator: which criteria are more important than others? If ‘adoption and popularity’ are the most important criterion, then clearly Spring Security is the framework to adopt for authorization and authentication. However, if ‘community engagement and maintainability’ are more important, then perhaps jGuard would be preferred over Spring Security. If vertical coverage is of paramount importance then JGuard or JAAS might be the top choice. Clearly there is no single, objective “best” framework, and any notion of best must be tempered by the evaluator’s goals (see Appendix A for an analysis of the different frameworks).

7. 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 particular frameworks, that is, there are not 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 any framework can be directly associated with tactics.

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 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.

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 methodology is justified in that the coverage metric is essential as it is the only one that focuses on the intent of the frameworks and their users.

Finally, not all of the data can be obtained equally. For example, it is not possible to differentiate between bug fixes and feature requests in some of the frameworks to calculate the community engagement metric.

8. Related work

Our work is unique due to the lack of research publications 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. [Ahamed2004] 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. [Jadhav2009] 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 in our work.

The work of [Hauge2009] 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 to 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 [Mohamed2007] 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. [Selle 2013].

9. Conclusions and Future Work

In this paper we have presented a methodology for selecting application frameworks focused on supporting the security quality attribute. Our methodology is intended to be practical, in the sense that a user can make a selection by applying objective evaluation criteria to the data, and this data can be collected in a relatively simple fashion.

This is the why our approach initially limited the number of evaluation criteria to the four categories presented in the paper: completeness, adoption and popularity, maintainability, and community engagement. Of the chosen criteria, completeness is the one that is the most subtle and least automatable--it requires the judgement of a human analyst. We decided, however, to keep this criterion as it is the only one that focuses specifically on the framework's and user's design intent.

Although we have tried to select our evaluation criteria based on information that is useful and insightful for practicing architects, reflecting their needs in developing real systems, we still need to validate our proposed method and its results with practicing architects to better understand whether they are willing to make selection decisions based on the information that our criteria provide. Nonetheless, we believe that our methodology represents a true contribution as it is far more detailed, holistic, and actionable than any prior research.

Future work includes industrial case studies and the identification of other criteria for which the data can be collected in a similar way to the criteria discussed in this paper. Finally, we plan to create a tool for practitioners that can provide them with a framework "profile" that evaluates a chosen set of frameworks across different criteria.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.
DM18-0778

References

[Ahamed2004] 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.

[Bass2012] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 3rd edition, Addison-Wesley, 2012.

[Cervantes2013] H. Cervantes, P. Velasco, and R. Kazman, "A Principled Way of Using Frameworks in Architectural Design," *IEEE Software*, 46–53, March/April 2013.

[Cervantes2016a] H. Cervantes, R. Kazman, J. Ryoo, D. Choi, D. Jang, “Architectural Approaches to Security: Four Case Studies”, *IEEE Computer*, 2016, to appear.

[Cervantes2016b] H. Cervantes, R. Kazman, *Designing Software Architectures: A Practical Approach,* Addison-Wesley, 2016.

[Fernandez2013] Fernandez-Buglioni, E., *Security Patterns in Practice: Designing Secure Architectures Using Software Patterns*, Wiley, 2013.

[Hauge2009] 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.

[Jadhav2009] A. Jadhav and R. Sonar, “Evaluating and Selecting Software Packages: A review”, *Journal of Information and Software Technology,* 51 555-563, 2009

[Mo2016] 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.

[Mohamed2007] 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.

[Piccioni2013] M. Piccioni, C. Furia, and B. Meyer, “An Empirical Study of API Usability”, *Proceedings of the International Symposium on Empirical Software Engineering and Measurement*, Baltimore, USA, October 2013.

[Potix2015] Potix Corp., Security Features of ZK Framework, Technical documentation of the ZK framework, 2013, http://books.zkoss.org/images/e/ea/ZK_Security_Report.pdf, last visited 11/11/15.

[Ryoo2015] Ryoo, J., Kazman, R., Anand, P., “Architectural Analysis for Security”, *IEEE Security and Privacy*, November/December 2015,13:6, 52-59.

[Scheller2015] T. Scheller, E. Kühn, “Automated measurement of API usability: The API Concepts Framework”, *Information and Software Technology*, Volume 61, May 2015, Pages 145–162.

[Selle2013] P. Selle, “13 Criteria for Evaluating Web Frameworks”, https://www.safaribooksonline.com/blog/2013/10/14/13-criteria-for-evaluating-web-frameworks/, 2013.

[Xiao2014] 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.

Biographies

Jungwoo Ryoo is a professor of Information Sciences and Technology (IST) at the Pennsylvania State University-Altoona. Ryoo is also a graduate/affiliate faculty member of the college of IST at Penn State. His research interests include information security and assurance,software engineering, and computer networking.

Junsung Cho is currently a graduate student with the Department of Computer Science and Engineering, Sungkyunkwan University, Suwon, Korea. His current research interests include usable security, mobile security, and security engineering.

Humberto Cervantes, Universidad Autonoma Metropolitana Iztapalapa, is a professor and researcher at Universidad Autónoma Metropolitana Itztapalapa. His current research interests include software architecture design, and he is an industry consultant and a certified Software Architecture Professional and ATAM Evaluator by the SEI.

Rick Kazman, is a professor at the University of Hawaii and research scientist at the Software Engineering Institute. He has created several influential methods and tools for architecture analysis, is author of more than two hundred peer-reviewed papers, and has coauthored several books, including Software Architecture in Practice, Third Edition.

Geumhwan Cho is currently a graduate student with the Department of Computer Science and Engineering, Sungkyunkwan University. His current research interests include usable security and mobile security.

Jina Kang is currently a developer and researcher in the National Security Research Institute, Daejeon, South Korea.

Hyoungshick Kim is an assistant professor with the Department of Software, Sungkyunkwan University. His current research interests include usable security and security engineering.

Appendix A: Security Framework Analysis Details

A.1 Frameworks Focusing on Cryptography

There are four security frameworks whose primary focus is cryptography. They include: JCE, Jasypt, Keyczar, and Bouncy Castle. Among these, the most comprehensive framework in its *vertical* coverage is Bouncy Castle. It acts as a JCE provider and implements all its API.

Compared to JCE and Bouncy Castle, the vertical coverage of Jasypt and Keyczar fall right in the middle. They offer a quick and easy cryptography implementations by providing user-friendly APIs for developers who do not need all the bells and whistles available through Bouncy Castle.

Therefore, regarding their horizontal coverage, we conclude that these frameworks fully support *encrypt data* and *verify message integrity* tactics while indirectly supporting derived security features such as: *identify and authenticate actors*. The indirect support means that developers still need to spend significant time in implementing their own design of these security features using the crypto functions instead of being able to rely on the native API available through the frameworks.

A.2 Frameworks Focusing on Authentication and Authorization

JGuard, OACC, PicketLink, and JAAS are authentication and authorization frameworks. JAAS is built into J2SE and provides interfaces that allow flexible authentication and authorization capabilities. JGuard builds on JAAS and supplies various implementations of its APIs.

For example, JAAS allows developers to create various login modules without fully implementing all of them, but JGuard provides a much more extensive and complete collection of login modules including: XMLLoginModule, OracleLoginModule, MySQLLoginModule, etc. It also provides additional Authorization Managers.

OACC is not JAAS-based but offers its own authentication and authorization architecture. It has a component similar to a login module, called AuthenicationProvidor, which by default provides simple, password-based authentication. It also make available special features such as identity delegation and multi-tenancy support as well as fully implemented data stores and Role-Based Access Control (RBAC).

PicketLink is another authentication and authorization framework independent of JAAS. It uses its own security models such as the concept of a partition and is geared towards supporting Java Enterprise Edition (EE) although some of its features support Java Standard Edition (SE). The main focus of PicketLink is identity management.

In terms of their vertical coverage, JAAS and JGuard are very comparable while OACC and PicketLink are providing their own unique security models and focus. That is, OACC

concentrates on the usability of its access control model while PicketLink emphasizes identity management.

A.3 Comprehensive Frameworks

These frameworks include OWASP Enterprise Security API (ESAPI), Spring Security, Apache Shiro, and HDIV. Among these, OWASP ESAPI, and HDIV are available only for web applications.

OWASP ESAPI is a *comprehensive* security framework designed to address all the OWASP top ten security concerns, which include: cross-site scripting (XSS), injection flaws, malicious file execution, insecure direct object reference, cross-site request forgery (CSRF), leakage and improper-error handling, broken authentication and sessions, insecure cryptographic storage, insecure communications, and failure to restrict URL access.

The security tactics covered by ESAPI are: identify actors, authenticate actors, encrypt data, limit access, validate input, detect intrusion, verify message integrity, and maintain audit trail.

When compared to ESAPI, Spring Security has a more limited horizontal coverage. Its coverage is much narrower in the core security area of: identify actors, authenticate actors, encrypt data, verify message integrity, and limit access.

Apache Shiro is similar to Spring security in its scope of horizontal coverage. The covered security tactics are: identify actors, authenticate actors, authorize actors, encrypt data, verify message integrity, and limit access.

The main difference between ESAPI and Shiro is that the former is intended for the advanced users of the built-in Java security models and their corresponding security APIs while the latter provides easy-to-use APIs for developers with limited Java security knowledge. Spring security also fits this description.

HDIV is a web application security framework that offers Runtime Application Self-Protection (RASP). It covers security tactics such as: validate input, detect intrusion, and maintain audit trail.