The Use of Security Tactics in Open Source Software Projects

Jungwoo Ryoo, *Member, IEEE*, Bryan Malone, Phillip A. Laplante, *Fellow, IEEE*, and Priya Anand

***Abstract*—Despite the best intentions of software architects, it is often the case that individual developers do not faithfully implement the original security design decisions. Such a scenario sometimes leads to a situation in which while an architect claims the use of a secure architecture in the form of some tactic, the corresponding source code does not support the claim. To bridge this gap, the first critical step is to verify whether the source code reflects at least some of the structural or behavioral features required for a tactic. In this study, we examine the extent of this discrepancy between an architect's vision of what security tactics need to be adopted in the software and the actual implementation. We accomplish this research goal by 1) exploring an architect's intention to use security tactics, 2) checking whether the tactic is manifested in the design, and finally 3) recovering the evidence of efforts to implement the design in the source code. To avoid limitations to accessing documentation and source code, we use open source projects to conduct our research.**

***Index Terms*—Open source software, security tactics, software architecture.**

I. Introduction

SOFTWARE security is becoming more important as the number and diversity of threats against software vulnerabilities grow [1]. One way of ensuring that software is secure is to adopt security countermeasures early on during a software development process [2], [3]. Security tactics have long been available to practitioners since they were introduced in [4]. However, many practitioners are unaware of the security tactics and rely on their intuition and informal learning from peers when trying to address the concerns associated with the security aspect of their software.

Therefore, we suspect that this informal and ad-hoc application of security principles and solutions to software design is more prevalent than the systematic approaches based on the adoption of tactics. As an initial step in ultimately testing our hypothesis, we reviewed the external documentation and the source code of 53 Open Source Software (OSS) projects that claim the use of security countermeasures [5]. We did not require the explicit mention of security tactics in the external documentation of these OSS projects when selecting them for our study subjects mainly because of our assumption that practitioners are using the security tactics implicitly, and some of the informal and coincidental use of security tactics will resemble the more formal practices reflected in an established security tactics hierarchy.

We believe that our research is significant because it establishes a baseline in terms of understanding the practical use of the security tactics. Understanding the degree of security tactics adoption in terms of consistency, correctness, and completeness lays a foundation for ultimately answering the deeper research question of whether security tactics are truly effective in addressing security vulnerabilities and threats taking advantage of them. Therefore, the immediate research goals of this work are to first quantify the extent of security tactics use in OSS projects and then to measure the gap between the claim and the actual use in the source code.

To accomplish these research goals, we developed a methodology to mine security-related artifacts from the source code, identified security tactics among the various security-related artifacts mined from the source code, developed a method to decide the extent of both formal and informal use of security tactics in practice, and finally decided whether the security tactics have been propagated in the source code at all.

II. Security Tactics

Tactics are architectural building blocks because they establish primitives for a software architecture. Tactics directly address quality attribute concerns such as availability, modifiability, security, performance, usability, and testability [4], [6], [8].

Ideally, selecting or creating a system architecture is the first stage in software development [9]. Extending beyond basic system functionality, quality concerns help reason about the architecture of the system [10]. The system should be developed with quality in mind from the onset of design rather than as an afterthought. However, no system should be developed for all quality attributes as this would be impractical because they serve different purposes that could conflict with each other [11]. In addition, these quality attributes are influenced by one another, and as such modifying one attribute will hardly be accomplished in isolation. For example, every change made to improve security quality attribute will often degrade performance as it causes more overhead in processing.

Efforts to build in "security" as a quality are ineffective when its existence cannot be tested [12]. Therefore, architectural design should begin with how one can test the qualities that the architecture claims to impart [13], [14]. Specifically, for security

Manuscript received September 02, 2014; revised March 02, 2015, June 29, 2015, September 03, 2015, and October 24, 2015; accepted November 10, 2015. Date of publication December 22, 2015; date of current version August 30, 2016. Associate Editor: W. E. Wong. (*Corresponding author: Jungwoo Ryoo.*)

J. Ryoo, P. A. Laplante, and P. Anand are with the College of Information Sciences and Technology, Penn State University, University Park, PA 16802 USA (e-mail: jryoo@psu.edu; axg36@ist.psu.edu).

B. Malone is with the Colorado Technical University, Colorado Springs, CO 80907 USA (e-mail: bmaloneco@gmail.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TR.2015.2500367

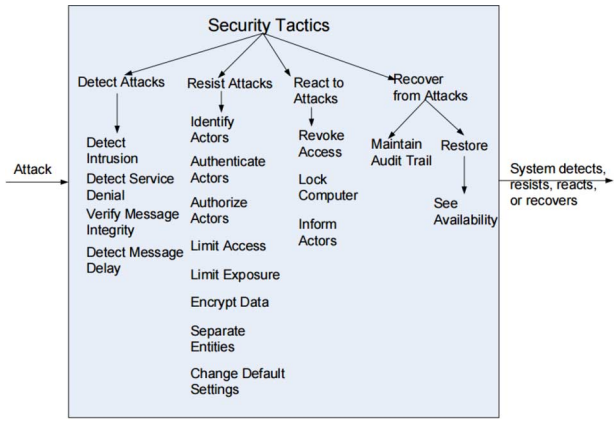


Fig. 1. Security tactics hierarchy.

purposes, testing model elements include a stimulus, its source, an environment, the target under attack, the desired response of the system, and the measure of this response. A stimulus is the actual attack or the attempt to break security. Secondarily, an artifact is the service or the data within a system. The attack can originate online/offline, inside/outside the network, or even if there is a firewall in place within the environment. A response is then determined based on whether the user is legitimate or not, and then appropriate measures are taken, using the enforcement of an audit trail where necessary. The response measure contends with recovery and survival depending on the severity of the attack [4].

Security for a system is defined by the following characteristics: non-repudiation (ensuring non-denial of messages being sent/received), confidentiality, integrity, availability, etc. But understanding these security requirements by no means implies that it will be easy to incorporate the corresponding controls in design. In this regard, security tactics fill the gap between security requirements and architectural design. Security tactics embody decisions about design, which address the security quality attribute-related concerns. Architectural patterns [13], [15], [14] can then adopt many of these tactics to enhance their security. Therefore, designing a secure system architecture involves making an assortment of related security decisions, and if any of those decisions shows a repeated behavior, then it has a potential to become a tactic. Specifically for securing systems, these tactics deal with resisting, detecting, and recovering from attacks [4] as shown in Fig. 1.

Next, evaluators of secure architectures will seek to ascertain the presence of these tactics in the source code.

III. Related Research

As discussed in Section II, there are a number of research works promoting the use of tactics during software development [4], [10], [16]–[18]. A majority of the existing research focuses on clearly defining what a tactic is [4], [16] and categorize them in a certain hierarchy to facilitate their adoption [4], [6], [18], [19]. Another important aspect of tactics research is to specify the tactics in various forms optimized for both human and machine consumption [20] as well as mining as many instances of tactics as possible [17], [18], [21], which could lead to more widespread adoption of tactics. There are also research efforts to provide methodologies for adopting tactics more properly and efficiently in a software development life cycle (SDLC) [10], [22], [23].

TABLE I
Sample Search Terms

auth	GRE	UDP
-authorize	hash	TCP
-authenticate	key	MD
AES	-Publickey	luks
DES	-Privatekey	protocol
Fish	Proxy	RFC
-blowfish	ELGamal	Hellman
-Twofish	DSA	secur
Kerberos	RSA	salt
Ciph	DSS	Sandbox
CBC	DER	VPN
credential	BER	VNC
cert	code	SHA
Crypt	-encode	SSH/X11
-encrypt	-decode	SSL
-decrypt	ftp	SSO or Realm
Rijndael	sftp	TLS
Anubis	pass	token
X.509	policy	utf
Khazad	PBE	Session
Escal of Privilege	POP	socket/port/connect
Multi-Term	http	

Very little has been done in terms of investigating how tactics are actually being used in real-life software systems, not to mention evaluating the effectiveness of their adoption. Therefore, our research is significant especially in a sense that it provides a seminal work that explores the degree of tactics adoption and provides a meaningful snapshot of the status quo. Part of the difficulties associated with this type of research is that the source code the target projects is not readily available, which is why we focus on the open source projects. In addition, the labor-intensive and manual nature of the work is another deterrence to the research in this field. Finally, the last challenge we would like to point out is the ambiguities in the use of terms for tactics, patterns, and architectural patterns. It is often the case that 'pattern' is an umbrella term used to refer to tactics and architectural patterns. Therefore, it is common to find the instances of tactics in the repositories of architectural patterns [13]–[15], [17], [24]–[33]. Although not as comprehensive as our study and not about tactics adoption, there was an attempt to provide more insights on the design pattern adoption [34]. In this case, Seen *et al.* examined the adoption of design patterns in major corporation such as AT&T, Siemens, etc. and offered their recommendation for the best design pattern adoption practices.

IV. Methodology

This study is purely based on observations. Therefore, there is no hypothesis to prove or disapprove. Instead, we developed a metric to first decide on a better indicator of security countermeasure implementations in source code (Section V-A and Section V-B). The second metric is used to measure the extent of the security tactics use in OSS projects (Section V-C).

A. Identification of Target Open Source Software (OSS)

The first step in the methodology was to identify candidate open source software applications for our analysis. Software

TABLE II
SECURITY KEYWORDS AND TACTICS MAPPING

S. No.	Key-words	Security Tactics									
		Resist Attacks						Detect Attacks	Recover from Attacks		React to Attacks
		Identify Actors	Authen--ticate	Autho--rize	Limit Access	Limit Exposure	Encrypt Data	Check Integrity	Maintain Audit	Establish Security	Inform Actors
1	auth	*	*	*							
2	auth--orize			*							
3	authen--ticate		*								
4	AES (Advanced Encryption System)						*				
5	DES (Data Encryption Standard)						*				
6	Fish						*				
7	blowfish						*				
8	twofish						*				
9	Ciph						*				
10	CBC (Cipher Block Chaining)						*				

project information was collected from Open source software (OSS) repositories, in particular SourceForge[1] and GitHub[2]. These two resources are the top providers for hosting OSS projects. One of the major criteria for including an OSS project in our study was whether our review of its external documentation revealed any claims of using security countermeasures. Using this criterion, we identified 53 OSS projects. Our hope was that some of these documented claims would lead us to discover the actual implementations of the security countermeasures in the form of tactics in the source code.

To provide a more randomized population sample, OSS resources were retrieved from the top-downloads for each week over the course of 32 weeks from http://sourceforge.com and http://github.com during the period of June 1, 2013 through January 11, 2014. While many of the applications were hosted on both of these websites, not all project files could be found on both sites.

B. Validation of Tactics Adoption in OSS Implementations

To validate whether their strong focus on security in the design documentation is propagated into the source code, we processed the candidate OSS source code with Structure 101 Studio[3], FileSeek[4] and 7-Zip[5] to extract class details. The search feature in Structure 101 was utilized to search for specific, security-related keywords such as "password" as shown in Table I.

Some of the security keywords were directly borrowed from the existing tactics hierarchy [4]. For instance, keywords such as "authenticate" and "authorize" directly correspond to the security tactics proposed by Bass *et al.* [4]. Other keywords such as X.509 and VPN derived from the security tactics. For example, both X.509 (a digital certificate standard) and Virtual Private Network (VPN: secure network communication standard) are security protocols used in networked applications. They are strongly related to the following tactics: identify actors, authenticate actors, limit access, and encrypt data.

When there are variations in the keyword (e.g., MD4, MD5, MD6, etc.), the invariable part of the keyword (i.e., MD) was used for the searches. The keywords beginning with hyphens indicate that they are the variations of the keyword without the hyphens directly above them. (e.g., auth vs.—authorize and—authenticate).

We ensured that the security keywords we identified have explicit mappings to the tactics hierarchy in Fig. 1 by creating a matrix shown in Table II. Note that we are showing only the top portion of the matrix due to the space limit.

Our searches aimed at file names, class names, and the internal documentation of source code. When multiple source code files were archived in a single file such as a .jar file, we used FileSeek and 7-Zip to extract individual files. Structure 101 offers behavior-preserving extraction in the form of class, collaboration and dependency diagrams. Therefore, once the security terms were identified, a deeper evaluation of the packages and classes was conducted in an attempt to determine the interactions of the classes. For example, Fig. 2 shows a package with the details of a credential class containing the cryptographic and MD5 features on the left-hand side of the window and the collaborators, java and javax on the right-hand side of the window. The bottom of the screen shows the dependency breakout and the use of org.mortbay.jetty.security framework and its constituent classes. In addition, since files typically indicate a structure in terms of classes, the view of

[1]http://sourceforge.net/

[2]https://github.com/

[3]http://structure101.com/

[4]http://www.fileseek.ca/

[5]http://www.7-zip.org/

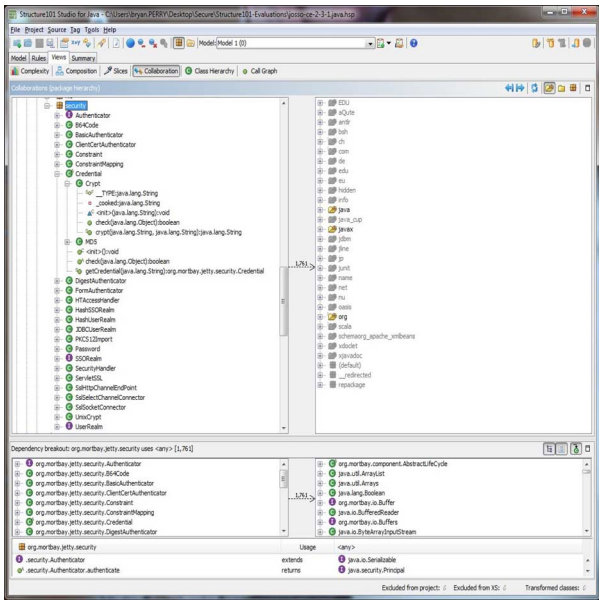


Fig. 2. Structure 101 application interface dependency breakout view.

the categories and class interactions exposes additional details about the system structure. This in-depth, manual review and evaluation of the source code allowed us to conclude if any of the claimed security features were implemented at all. Measuring the degree of implementation (e.g., fully or partially) was out of the scope of this paper.

V. Findings

This section describes all the major findings of our study. We first present our 'threats to validity' analysis results with respect to our file name search method used to locate security artifacts such as security-relevant files, classes and their relationships.

A. Threats to Validity Analysis

One of the threats to validity is that the file name searches using the security keywords do not always lead to the discovery of security code. For example, 17 of the 53 OSS projects (34%) we studied revealed no security code at all when their file names indicated the use of security countermeasures. The reverse is also true. It is possible that files with no apparent association with security countermeasures in their names contain security code. Fig. 3 shows the ratio of the actual number of security classes revealed after further inspections and the total number of classes inspected as a result of security keyword matches in the searches we conducted in this study. We use this metric (i.e., the ratio) to measure the effectiveness of file names and their alternative as indicators of security countermeasure implementations. Therefore, the metric can be formally denoted as:

$$E_i = \frac{I_s}{I_k}$$

where E_i stands for the effectiveness of an indicator, and I_s and I_k stand for the number of keyword matches actually leading to the security countermeasure code and the number of the keyword matches respectively when using the indicator. Another

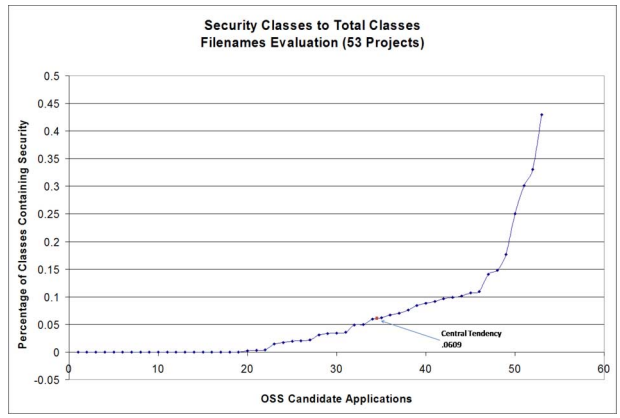


Fig. 3. File names leading to security code.

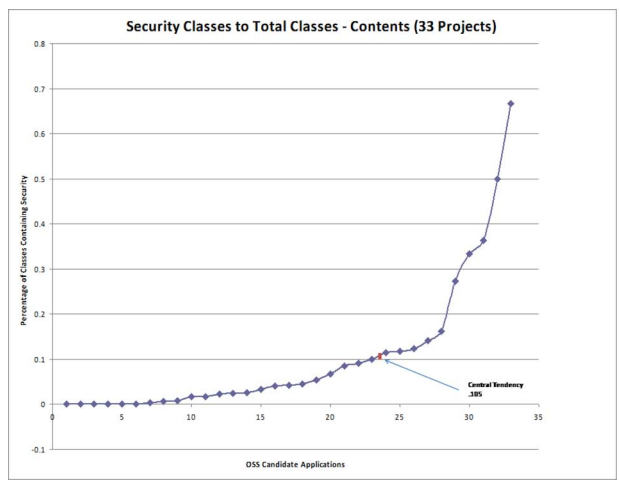


Fig. 4. Security classes to total classes.

threat to validity lies in the manual nature of our code reviews, which could be error-prone.

B. Search Results and Analysis

1) Security Countermeasure Adoption in the Code: As discussed in Section IV, we conducted searches of security keywords to locate the areas of source code, containing the evidence of security countermeasure implementations. After considering the threats to validity due to the misleading file names, we decided to use class names for our security keyword searches. Therefore, the indicator, I we used in this study is the class names. We had to reduced our sample size to 33 to remove outliers that had either too many or too few keyword matches due to the more labor-intensive nature of the class name searches. In our reduced sampling, 27 of 33 projects studied (81.9) had classes with security intent. For this reduced sample, there is an increase in confidence with the central tendency (average) of E_i increasing by 4.41% to 10.5% (Fig. 4), thus providing a better reflection of security classes found in the content over the filename recovery.

Based on this result, we can conclude that a majority of the OSS projects (81.8%) reviewed in this study do have classes implementing security countermeasures as they claim in their external documentation. However, the remaining question is how many of these classes implement security tactics and other

TABLE III
SECURITY CLASS DISTRIBUTION PER TACTIC CATEGORY

Percentage of Total Classes by Classification			
Total Classes containing Security in Sample Population 9811			
			Tactic – Resist Attacks
Keywords	**Classes**	**Percentage**	**Classification**
Authenticate	1343	13.7%	Authenticate Actors
Authorize	70	0.7%	Authorize Actors
Auth-unclear	166	1.7%	Authenticate Actors Authorize Actors
Transport	343	3.5%	Identify Actors
Connection	1182	12.0%	Encrypt Data Check Integrity
Cryptography- Standards	52	0.5%	Encrypt Data
Encryption	3219	32.8%	Encrypt Data
Encode	1244	12.7%	Encrypt Data
Protocol	996	10.2%	Identify Actors
Escalation of Privilege	9	0.1%	Authorize Actors
Request For Comments	43	0.4%	Identify Actors Authenticate Actors Authorize Actors Encrypt Data Check Integrity
Sandbox	48	0.5%	Limit Access
Security	353	3.6%	All-encompassing
Single Sign On	18	0.2%	Authenticate Actor
Standards	711	7.2%	Establish Security Standards
Network Monitoring	14	0.1%	Maintain Audit Trail

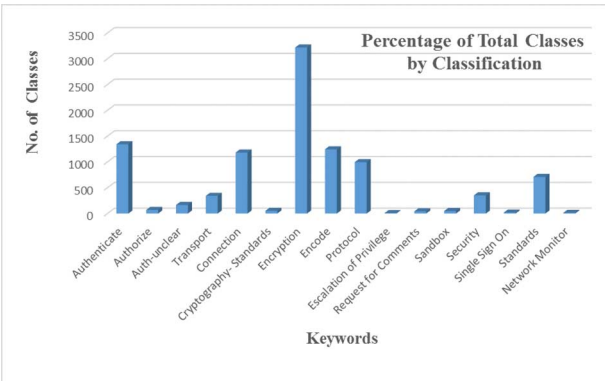


Fig. 5. Percentage of total classes by classification.

forms of security design concepts such as security patterns. Another important research question is how these security tactics and patterns are used in the OSS projects. For example, the distribution of the adoption of known security tactics and patterns may provide some useful insights. In this paper, we focus on the investigation of the former research question.

C. Security Tactics Adoption in the Code

Table III and Fig. 5 shows the distribution of security classes in the sample population of the total 9811 classes we discovered through the keyword searches as part of this study.

We discuss each entry of Table III in the remainder of this section.

1) Authenticate: Authentication ensures that a user is who they claim to be. From a security tactics standpoint, authentication seems to be a standard for many of the applications for users who conduct digital identification when interacting with network and system resources. Certificates, which are similar to authentication, use asymmetric encryption to identify and verify that the sender of the message (rather than the receiver) is who they claim to be and in this regard are like a passport. Tokens can also represent a form of a certified entry, or unique identification code used to confirm that a session, packet or message is factual. Certificates are digitally signed and intended to provide nonrepudiation. One example is found in the use of a Certificate Authority (CA) issuing a digital certificate that contains a public key. The measure of classes signifies that there is substantial use of security tactics for user identification, certificates and tokens as it reflects 13.7% of the total classes in the population sample (Table III). This category ties specifically to the search for authentication actors by identifying roles, groups and privileges in tactics resisting attacks.

2) Authorize: Authorization grants users access to system features and resources. From Table III, there are 0.7% of classes controlling users' access rights. This is a seemingly small number given the nature of the task. This category ties specifically to the search for authorized actors by identifying their roles, groups and privileges in the resisting attacks tactics category. The auth category was used for classes that defy a clear division between authentication and authorization. However, use of auth represents 1.7% of the total classes in the population sample. While the combined use of the authentication and authorization tactics seems to have related interactions, there is a substantial difference in the implementation of these security tactics. There is no direct tie to a tactic category other than as a function of resisting tactics.

TABLE IV
SAMPLE SECURITY KEYWORD CLASS SEARCH RESULTS

Sample Classes by Classification			
	Class 1	**Class 2**	**Class 3**
Authenticate	Credential.class	KerberosPrincipal.class	Credential$MD5.class
Authorize	Authorization.class	AuthorizeCallback.class	AuthorizationDataEntry.class
Auth-unclear	AuthPolicy.class	AuthScope.class	AuthState.class
Transport	nsISSLStatus.class	nsISSLErrorListener.class	socat-openssltunnel.html
Connection	ui_openssl.c	PortValidator.class	InetSocketAddress.h
Cryptography -Standards	JCEBlockCipher$ PBEWithSHA1AndDES.class	CipherHelper InputStream.class	BrokenJCEBlockCipher $OldPBEWithSHAAndTwofish.class
Encryption	RSAKeyPair.class	ElGamalKey.class	P11RSACipher.class
Encode	CodeSigner.class	RSASignature$SHA1.class	ParametersWithSalt.class
Protocol	ftp_browsertest.cc	pseudotcp_unittest.cc	SSLProtocolSocketFactory.class
Escalation of Privilege	SmapUtil.class	SmapGenerator.class	SmapUtil$PreScanVisitor.class
Request for Comments	RFC1522Codec.class	RFC2109Spec.class	RFC2617Scheme.class
Sandbox	sandbox_policy.h	sandbox_factory.h	nsIXPCComponents _utils_Sandbox.class
Single Sign On	SSORealm.class	HashSSORealm.class	FormAuthenticator$SSOSignoff.class
Security	SecuritySupport.class	SecurityClassLoad.class	SecurityHandler$NotChecked.class
Standards	libipt_quota.so	LayoutFocusTraversalPolicy.h	nsIUTF8ConverterService.class
Network Monitor	snmpcl.def	snmpincl.def	snmpsmir.def

TABLE V
SPNEGO CLASS NAMES AND THEIR FUNCTIONS

No.	Class Name	Intended Function
1	SpnegoAuthenticator	Handles basic authentication
2	SpnegoAuthScheme	Handles basic authorization and negotiate schemes
3	SpnegoFilterConfig	Class that applies/enforces web.xml init params
4	SpnegoHttpFilter	Http Servlet Filter that provides servlet authentication
5	SpnegoHttpFilterConstant	Defines constants and parameter names that are used in HTTP request headers.
6	SpnegoHttpServletRequest	Wraps ServletRequest so that the project can handle its own principal and authentication types.
7	SpnegoHttpServletResponse	Adds a capability to track/determine if the HTTP Status code has been set.
8	SpnegoHttpURLConnection	To connect to a protected HTTP server
9	SpnegoPrincipal	To encapsulate a KerberosPrincipal
10	SpnegoProvider	To get control over integrity, confidentiality and authentication
11	SpnegoSOAPConnection	To make SOAP calls to a protected SOAP Web Service

3) Transport: Transportation of data packets, often referred to as datagrams, across channels requires some form of protection as there is no method for ensuring the error-free delivery of those packets. Transportation features ensure the delivery of messages for key applications such as web applications, instant messaging and Voice-over-IP (VoIP) solutions, using datagram services as the backbone delivery method. The transport category contains only 3.5% of the total classes in the population sample. This category specifically ties to the search for the "identify actors" tactic in the resisting attacks tactics category since it requires the source and destination addresses for transmitting packets.

4) Connection: Connections are an assurance between two devices or nodes on the network. Connections, or sockets and proxies fall into the category of ensuring error-free data transmission. To verify the connection between two nodes, certificates issued from VeriSign, Symantec, GoDaddy and others continues to rise as a means to ensure the connection is not only trusted but also validated especially in terms of message integrity. Use of connections was the fourth most numerous class in the study containing 12.0% of the total classes in the population sample. This category specifically ties to the search for the "encrypt data" tactic in the resisting attacks tactics category. In addition, this class category demands an addition of a new tactic category, which is "check integrity," which is a crucial security feature not yet documented in the existing tactics hierarchies.

5) Cryptography-Standards: Cryptography standards seek to provide a foundation for defining the use of necessary standards to ensure cryptography works through consistent data encryption. In Table III, "standards" refer to cryptographic hash functions used to verify data integrity. Standards in cryptography produced 0.5% of the total classes in the population sample. This category specifically ties to the search for the "encrypted data" tactic in resisting attacks tactics.

6) Encrypt: "Cryptography" is, however, identified as encryption but encompasses decryption and various ciphers whose algorithms are used to make encryption work. While the standards do not produce much in terms of the sample population, the use of cryptography itself is quite substantial in its representation in the sample population. Cryptography represents the largest component in security classes with 32.8% of the total classes in the population sample. This category specifically ties to the search for the "encrypt data" tactic in the resisting attacks tactics category.

7) Encode: The goal of encoding is to represent the data, message, or packet in some form to be consumed by a receiver in data transfer. The goal of decoding is to remove the encoding

TABLE VI
SECURITY KEYWORDS AND TACTICS IMPLEMENTATION

No.	Class Name	Security Keywords	Security Tactics Used
1	SpnegoAuthenticator	Yes	Yes
2	SpnegoAuthScheme	Yes	Yes
3	SpnegoFilterConfig	No	No
4	SpnegoHttpFilter	Yes	Yes
5	SpnegoHttpFilterConstant	Yes	No
6	SpnegoHttpServletRequest	Yes	No
7	SpnegoHttpServletResponse	Yes	No
8	SpnegoHttpURLConnection	Yes	No
9	SpnegoPrincipal	No	Yes
10	SpnegoProvider	No	No
11	SpnegoSOAPConnection	No	No

scheme. Encoding is often confused with encryption, which is why we decided to still add this keyword to our search. During the compiling of the search results, key prefixes such as "code," "encode," and "decode" were combined as it is not clear by the class names what the classes actually contained. Encode was the third largest component in the findings with 12.7% of the total classes in the population sample. This category specifically ties to the search for the "encrypt data" tactic in the resisting attack tactics category.

8) Protocol: Protocols establish rules of "behavior" for messages and seek to define principles of session connection, network standards, reference model conformance, tunneling delivery assurance and more. Protocols provide the essence of structure for both the hand shaking between clients and servers and for data transmission, connection and interaction between the clients and servers, either through application layer content delivery or other network layers. Protocols reflected 10.2% of the total classes in the population sample. This category specifically ties to the search for the "identify actors" tactic in the resisting attacks tactics category since it involves identifying the involved parties especially in the security context.

9) Escalation of Privilege: Privilege escalation is a vulnerability granting access rights at a higher level than the user is assigned upon being authorized. While this designation comprised a relatively low percentage of the overall findings, Escalation of Privilege actually appeared as classes in the Zocalo application (http://zocalo.sourceforge.net/javadoc/) and is further detailed in design and implementation documentation at its website as a means to prevent privilege escalation. Standards in general seem to reflect a higher level of classes representing this category. While the Escalation of Privilege countermeasures reflect relatively low presence of security measures, the escalation of privilege attacks are well-known, and they can have far-reaching implications for any future research directive related to security tactics. Escalation of Privilege reflected 0.1% of the total classes in the population sample. Classes in this category are specifically tied to the search for the "authorized actors" tactic in the resisting attacks tactics category.

10) Request for Comments (RFC): This set of classes pertain to all methods, behaviors, innovations, and research, which apply to the Internet and networked systems including security countermeasures. RFCs are managed by the Internet Engineering Task Force (IETF). Research on RFCs demonstrates limited design emphasis in the sample population. For example, the specific class definitions for RFC 6101 address network characteristics in the use of Secure Socket Layer (SSL) standards. RFCs comprised 0.4% of the total classes. While constituting a considerably low percentage of total classes, RFCs are used for every aspect of security implementations in Internet-based or networked systems. This category has multiple ties to the security tactics classifications, and the class implementation of an RFC that we found was tied to the "identify," "authenticate," "authorize," and "encrypt" tactics.

11) Sandbox: Sandbox is a security measure created in Java development, which contains a set of rules used to prevent applets from performing many different undesirable activities such as accessing certain designated resources. A Java applet is embedded in the webpage, and Java security relies on the byte code verifier, class loader, and the security manager to restrict file system access, network access, and access to browsers internal features. Three applications use the Sandbox concept: Java Sandbox, Chromium, and JOSSO. Sandboxes comprised 0.5% of the total classes in the population sample. This category specifically ties to the search for the "limit access tactic" in the resisting attacks tactics category.

12) Security: Security is an all-encompassing term that is self-defining. Security measures seek to define who or what can have access to system resources. The prefix "secur" and the term security produce marginally significant contributions but require additional exploration. Future research in the use of security needs clarification. Security comprised 3.6% of the total classes in the population sample. This category had no ties to the search for any particular security tactics as these could be detecting, resisting, reacting to or recovering from attacks.

13) Single Sign on: Another significant revelation reflecting relatively low measures of security and a known weakness is found in the use of Single Sign-On (SSO). Additional research in this realm could yield a significant contribution. SSOs inquire an authenticator of if any SSO exists for a given request. In the event of successful authentication, the credentials are returned. Single Sign-On represented 0.2% of the total classes in the population sample. This category specifically relies upon the search for the "authenticate actors" tactic in the resisting attacks tactics category.

14) Standards: Security standards or "establish security standards" should be considered as another candidate tactic. Standards addressing security policies are mandatory requirements based on a community consensus for protecting the system against threats or vulnerabilities. Character representation schemes can be substantially more complex with additional security standards if users seek to exploit any vulnerabilities when they are expected to behave benignly. Ultimately, uniform conformance to security standards blocks opportunities to intentionally break processes and procedures. While significantly harder to track, standards nevertheless comprised 7.2% of the total classes in the population sample. This category has no specific ties to the existing security tactics classifications.

15) Network Monitoring: This classification within our taxonomy is the set of classes which are responsible for logging and monitoring. For example, the network monitoring scheme could be the Simple Network Management Protocol (SNMP)

TABLE VII
PROJECTS AND BRIEF DESCRIPTIONS

Cnt	Project	Description (brief)
1	amule 2.3.1	multi-platform ed2k client
2	Ares Galaxy	a built in audio/video viewer
3	awstats 7.0	web server logfile analyzer shows web stats
4	chromium	more stable open-source web browser
5	codeblocks	C, C++ and Fortran IDE
6	Cryptsetup 1.6.3	utility used to conveniently setup disk encryption
7	ettercap 0.7.6	multipurpose sniffer/interceptor/logger - LAN
8	Filezilla	cross-platform graphical FTP, FTPS and SFTP client
9	giFT	Improve use of peer-to-peer file-sharing networks
10	gparted-live 0.16.1-1	linux disk partitioning tool
11	HTTPLoris 3.2	python plug-in- simulate dDOS attacks protocol agnostic app layer
12	jasypt 1.9.1 Java Simplified Encryption	add basic encryption capabilities
13	Java Mechanic 3.0	loads other Java applications and applets
14	Java Sandbox	a sandbox for Java code
15	Jess	uses visitor pattern to find security bugs
16	josso	Internet Single Sign-On (FSSO) solution
17	jQuery JavaScript	query-based source code browser Eclipse plugin
18	jsch-0.1.49 Java Secure Channel	a pure Java implementation of SSH2
19	JSignPdf_setup_1.5.1_wjre	adds digital signatures to PDF
20	Libdnet-1.11 (dnet-1.11.win32-py2.4	portable interface low-level networking routines
21	mcrypt-2.6.7-win32	encrypt files/data streams w/o cryptographers
22	Metasploitable2-Linux	intentionally vulnerable Linux virtual machine
23	Milk-0.25	security source code assess tool W/Orizon API
24	mingw w64 v 3.0.0	runtime, headers and libs for develop 32/64bit
25	One Jar	Java app puts depend into single exec Jar
26	ophcrack-vista-livecd-3.6.0	Windows password cracker
27	OpenSSO CertificateLogin 1.0	allows users to authenticate themselves via their certificates
28	OWASP Mantra Janus-MoCen 0.3 Alpha	Browser based Security Framework (on Chromium)
29	OWASP Mutillidae	deliberately vulnerable web-application a target for web-security
30	OWASP WebGoat	deliberately insecure web application
31	OWASP Pantera	web application analysis engine works with penetration testing
32	Paradroid Sourcefile13	clone of the classic Commodore 64 game Paradroid
33	PasswordSafeSWT-0-92.beta	stores your passwords in an encrypted file
34	phpMyAdmin 4.0.9	administration of MySQL over the web
35	pmd bin 5.0.5	source code analyzer finds common prog flaws
36	Poisoned 5191	Cocoa Interface to the giFT Daemon
37	Pcre 8.33	Perl-compatible regular expression library
38	PWGen-2.3.1-Setup	cryptographically-secure passwords or passphrases
39	Python 2.4	programming development language
40	Screen Detroyer	first person shooter game allowing destroying objects on desktop
41	ScrolloutF1	email gateway- Secure email servers easy
42	Sleuth Kit 4.1.2	view allocated/deleted data from File systems
43	Spnego	alternative library for authenticating clients
44	swigwin 2.0.11	reads annot C hdr files & creates wrapper code
45	Turn.js Master	Make a flip book effect with HTML5
46	UDP Unicorn	UDP flooding/DoS utility with multithreading
47	VLC 2.0.6	media playback and streaming software
48	Vuze 4604	P2P file sharing client
49	WinMerge 2.14.0	display and merging files/directories
50	Xbrlapi 6.2	Java API for XBR
51	zen cart 1.1.4d	securi online shopping cart
52	zk bin 6.5.4	Ajax+Mobile Java Web framework
53	zocalo-PM-2011-2-bin	toolkit for building Prediction Markets

which performs network evaluations and determines what activities are taking place while providing service-based solutions to end-users. SNMP is also known for adding unnecessary overhead to the network, but it is required for troubleshooting situational network problems. These problems can consist of malfunctioning network devices, etc. In general, SNMP was not utilized on a level which would suggest significance and is used typically as the first keyword as a determinant for the classification direction for which classes will be placed. Only two applications used the network monitoring category: JOSSO and MingW. Network monitoring represented 0.1% of the total classes in the population sample. This category has direct ties to the "maintain audit trail" tactics.

While not all-inclusive, many of the findings in the class name search category (Table III) can be associated with the "encrypt data" tactic for the resisting attacks tactics category. Four of the sixteen classifications reflected the use of encrypted data resisting attacks, thus comprising 25.0% of the total classification findings. This revelation points to a heavy emphasis in the use of encrypted data for the "resist attacks" tactics category. Further,

in the analysis of the defined search terms, 25 of 60 contained encryption solutions along with another two addressing cryptographic standards, thus comprising 41.67% of the search terms.

Cryptography is not a comprehensive solution to the security problem, and this focus on encryption indicates a big gap in the current adoption of security tactics in OSS.

VI. Case Study

To demonstrate how our proposed methodology works more clearly, we provide a case study in this section. The open source project we selected for this case study is SPNEGO: http://spnego.sourceforge.net/, which provides an alternative library for application servers to use to authenticate their clients. SPNEGO clearly states in its design document that it addresses the *Authenticate Actors* and *Authorize Actors* tactics whose implementation has a total of 11 classes. Based on the SPNEGO project documentation, we list its class names and their intended functions in Table V.

We manually inspected the SPNEGO source code to verify whether each class actually implemented the security tactic whose use was claimed in the design documentation. Table VI summarizes the result of our manual inspection.

This case study clearly demonstrates how we identify the extent of security tactics use in an open source project. While some class names imply the use of security tactics in them, only four out of the seven classes actually implement the tactics. This ratio of the security classes to the total classes for the SPNEGO project is reflected in Fig. 4. This case study also helps reinforce our belief that security tactics are not completely implemented as either hinted in their class names or as intended during the design phase.

VII. Conclusion

In this paper we investigated the degree of security tactics adoption in the OSS projects. We conducted our investigation at both design and implementation levels. The results are mixed. While a majority of the OSS projects adopt security tactics in both design and implementation, the scope of the adoption was quite limited. That is, the adoption only focused on a certain set of tactics such as "encrypt data" in the tactics hierarchy. Other tactics were very rarely used or not used at all. This presents an opportunity for the OSS community to explore other less heavily used security tactics and consider them for adoption. We believe that this unbalanced adoption of security tactics is mainly due to the lack of knowledge in security tactics rather than intentional avoidance of using certain security tactics.

We also realize that there are several security tactics candidates that could be incorporated into the existing tactics hierarchy. The security tactics candidates discovered during our search for the evidence of security countermeasures in the source code are especially valuable because they are what is being used by practitioners. Another reason for their significance is that the formation of the tactics hierarchy is also partially dependent on this community process of proposing security tactics and adopting them by the practitioners. Therefore, future research should identify more candidate security tactics used in practice and find ways to systematically incorporate them in the existing security tactics hierarchy. Further research is also necessary to evaluate the effectiveness of the newly discovered security tactics when they are incorporated into a tactics hierarchy.

References

[1] G. McGraw, *Software Security: Building Security*. Boston, MA, USA: Addison-Wesley Professional, 2006.

[2] J. H. Allen, S. J. Barnum, R. J. Ellison, G. McGraw, and N. R. Mead, *Software Security Engineering: A Guide for Project Managers*, 1st ed. Boston, MA, USA: Addison-Wesley Professional, 2008.

[3] R. Augliere, Software Security Total Risk Management, Jul. 08, 2012 [Online]. Available: http:www.ittoday.info/Articles/Software_Security_Total_Risk_Management.htm

[4] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 3rd ed. Boston, MA, USA: Addison-Wesley Professional, 2012.

[5] OWASP Secure Coding Practices—Quick Reference Guide-OWASP, Jun. 13, 2014 [Online]. Available: https://www.owasp.org/index.php/OWASP_Secure_Coding_Practices_-_Quick_Reference_Guide

[6] J. Ryoo, "Revising a security tactics hierarchy through decomposition, reclassification, and derivation," in *Proc. 2012 IEEE 6th Int. Conf. Software Security and Reliability Companion (SERE-C)*, Gaithersburg, MD, USA, 2012, pp. 85–91.

[7] J. Ryoo, P. Laplante, and R. Kazman, "A methodology for mining security tactics from security patterns," in *Proc. 2010 43rd Hawaii Int. Conf. System Sciences*, Honolulu, HI, USA, 2010, pp. 1–5.

[8] J. Ryoo, P. Laplante, and R. Kazman, "In search of architectural patterns for software security," *Computer*, vol. 42, no. 6, pp. 98–100, Jun. 2009.

[9] F. Bachmann, L. Bass, and M. Klein, "Moving from quality attribute requirements to architectural decisions," in *Proc. 2nd Int. SofTware Requirements to Architectures Workshop (STRAW'03)*, 2003, p. 122.

[10] S. Kim, D.-K. Kim, L. Lu, and S. Park, "Quality-driven architecture development using architectural tactics," *J. Syst. Softw.*, vol. 82, no. 8, pp. 1211–1231, 2009.

[11] R. Kazman, *Evaluating Software Architectures: Methods and Case Studies*, 1st ed. Boston, MA, USA: Addison-Wesley Professional, 2001.

[12] G. McGraw, "Software assurance for security," *IEEE Comput.*, vol. 32, no. 4, pp. 103–105, Apr. 1999.

[13] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1st ed. Boston, MA, USA: Addison-Wesley Professional, 1994.

[14] E. Fernandez-Buglioni, *Security Patterns in Practice: Designing Secure Architectures Using Software Patterns*, 1 ed. Chichester, U.K.: Wiley, 2013.

[15] B. Blakley and C. Heath, "Security design patterns," *The Open Group,* Apr. 2004 [Online]. Available: http://www.opengroup.org/pubs/catalog/g031.htm

[16] N. B. Harrison and P. Avgeriou, "How do architecture patterns and tactics interact? A model and annotation," *J. Syst. Softw.*, vol. 83, no. 10, pp. 1735–1758, Oct. 2010.

[17] J. Ryoo, P. Laplante, and R. Kazman, "A methodology for mining security tactics from security patterns," in *Proc. Hawaii Int. Conf. System Sciences*, Los Alamitos, CA, USA, 2010, pp. 1–5.

[18] J. Scott and R. Kazman, *Realizing and Refining Architectural Tactics: Availability*. : C.-M. U. P. P. S. E. INST, 2009.

[19] L. Pavlic, M. Heričo, and V. Podgorelec, "Improving design pattern adoption with ontology-based design pattern repository," in *Proc. 30th Int. Conf. Inf. Technol. Interfaces, 2008 (ITI 2008)*, 2008, pp. 649–654.

[20] A. Wyeth and C. Zhang, "Formal specification of software architecture security tactics," *SEKE*, pp. 172–175, 2010.

[21] F. Bachmann, L. Bass, and M. Klein, "Deriving architectural tactics: A step toward methodical architectural design," in *CMU/SEI-2003-TR-004, ADA413644*, Pittsburgh, PA, USA, Mar. 2003.

[22] S. Kim, D.-K. Kim, L. Lu, and S.-Y. Park, "A tactic-based approach to embodying non-functional requirements into software architectures," in *Proc. 2008 12th Int. IEEE Enterprise Distributed Object Computing Conf.*, Munich, Germany, 2008, pp. 139–148.

[23] M. Mirakhorli, P. Mäder, and J. Cleland-Huang, "Variability points and design pattern usage in architectural tactics," in *Proc.e ACM SIGSOFT 20th Int. Symp. Found. of Softw. Eng.*, New York, NY, USA, 2012, pp. 52:1–52:11.

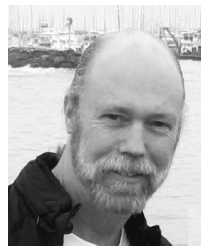
[24] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture Volume 1: A System of Patterns*, 1st ed. New York, NY, USA: Wiley, 1996.

[25] C. Dougherty, K. Sayre, R. Seacord, D. Svoboda, and K. Togashi, "Secure design patterns," *SEI, TR-010-2009*, Oct. 2009.
[26] M. Hafiz, P. Adamczyk, and R. E. Johnson, "Organizing security patterns," *IEEE Softw.*, vol. 24, no. 4, pp. 52–60, 2007.
[27] M. Hafiz, P. Adamczyk, and R. E. Johnson, "Growing a pattern language (for security)," in *Proc. ACM Int. Symp. New Ideas, New Paradigms, and Reflections on Programming and Software*, New York, NY, USA, 2012, pp. 139–158.
[28] C. Steel, R. Nagappan, and R. Lai, *Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management*, 1st ed. Englewood Cliffs, NJ, USA: Prentice Hall, 2012.
[29] M. Schumacher, *Security Engineering With Patterns: Origins, Theoretical Models, and New Applications*, 1st ed. New York, NY, USA: Springer, 2003.
[30] M. Schumacher, *Security Patterns: Integrating Security and Systems Engineering*. New York, NY, USA: Wiley, 2006.
[31] H. W. Nobukazu Yoshioka, "A survey on security patterns," *Prog. Inf.*, no. 5, 2008.
[32] D. G. Rosado, C. Gutierrez, E. Fernandez-Medina, and M. Piattini, "A study of security architectural patterns," in *Proc. 1st Int. Conf. Availability, Rel. and Security, 2006 (ARES 2006)*, 2006, p. 8.
[33] T. Heyman, K. Yskout, R. Scandariato, and W. Joosen, "An analysis of the security patterns landscape," in *Proc. 3rd Int. Workshop Softw. Eng. for Secure Syst.*, Washington, DC, USA, 2007, p. 3.
[34] M. Seen, P. Taylor, and M. Dick, "Applying a crystal ball to design pattern adoption," in *Proc. 33rd Int. Conf. Technol. of Object-Oriented Languages, 2000 (Tools 33)*, 2000, pp. 443–454.

Jungwoo Ryoo (M'04) received the Ph.D. degree in computer science from the University of Kansas, Lawrence, KS, USA, in 2005.

He is an Associate Professor of information sciences and technology at the Pennsylvania State University, Altoona, PA, USA. His research interests include information assurance and security, software engineering, and computer networking.

Prof. Ryoo is a member the ACM and a technical editor of the IEEE COMMUNICATIONS MAGAZINE.



Bryan Malone is the Director of Information Technology at Perry Supply Company, Albuquerque, NM, USA.

Philip A. Laplante (M'86–SM'90–F'08) received the B.S. degree in systems planning and management, the M.Eng. degree in electrical engineering, and the Ph.D. degree in computer science from the Stevens Institute of Technology, Hoboken, NJ, USA, in 1983, 1986, and 1990, respectively, and the M.B.A. degree from the University of Colorado, Colorado Springs, CO, USA, in 1999.

He is a Professor of software and systems engineering with Pennsylvania State University, Malvern, PA, USA. His research interests include real-time systems, real-time image processing, safety critical software systems, and software quality.

Priya Anand is working toward the Ph.D. degree at the Pennsylvania State University, Altoona, PA, USA.

She is a part-time instructor at the Pennsylvania State University, Altoona. Her research interests include software architecture and software security.