In Proceedings of 51st Hawaii International Conference on System Sciences (HICSS), Hawaii, US, 2018

An Empirical Study of Security Issues Posted in Open Source Projects

Mansooreh Zahedi
IT University of Copenhagen
mzah@itu.dk

Muhammad Ali Babar
University of Adelaide
ali.babar@adelaide.edu.au

Christoph Treude
University of Adelaide
christoph.treude@adelaide.edu.au

Abstract

When developers gain thorough understanding and knowledge of software security, they can produce more secure software. This study aims at empirically identifying and understanding the security issues posted on a random sample of GitHub repositories. We tried to understand the presence of security issues and their key themes and topics. We applied a mixed-methods approach, combining topic modeling techniques and qualitative analysis. Our findings have revealed that a) the rate of security-related issues was rather small (approx. 3% of all issues), b) the majority of the security issues were related to identity management and cryptography topics. We present 7 high-level themes of problems that developers face in implementing security features.

1. Introduction

With a growth in connectivity of systems and services to the Internet, security of software systems is increasingly becoming important. Security vulnerabilities can expose a system to attackers for stealing sensitive data and performing malicious activities that sometimes have tremendous impact. The most recent example is the "WannaCry" attack in May 2017, which targeted more than 90 countries and infected internal systems of organizations, e.g., hospitals in the UK. Therefore, different security mechanisms, guidelines and tools are continuously being provided with the purpose of decreasing software security risks. It is found that scarcity of security professionals and developers' lack of knowledge of secure coding are major concerns particularly for web app development [1]. This motivates researchers (e.g., [2]) to investigate solutions to improve security skills of developers. The lack of security professionals can be due to ineffective methods (e.g., relying on apprenticeship) that are normally used for sharing security knowledge [3]. As a result, some initiatives (e.g., [3], [4]) are taken to systematically organize software security knowledge to be used by researchers and practitioners in this domain. There also exist well-known security dictionaries and catalogues (e.g., Common Vulnerabilities and Exposures - CVE) that are maintained with categorized information about vulnerabilities. Despite their popularity, these catalogues typically have a very complicated structure that makes them difficult to use.

In this paper, we explore the presence and key themes of security issues in GitHub repositories. We define a *"security issue"* as: a posted issue on GitHub that contains a security-related aspect. Taking a bottom-up approach, we aim to understand security-related topics and themes emerging from issues that software developers raised. We discuss that this tactic could help us realize the difficulties that developers face in this regard and identify the required knowledge areas. Our study aimed at exploring the following research questions:

RQ1: What is the rate of security issues posted in open source projects?

RQ2: What are the most frequent security keywords appearing in the issues of open source projects?

RQ3: What are the main themes and topics of security issues in open source projects?

We used a mixed-methods approach, combining topic modeling and qualitative analysis to answer our research questions. We collected the issues from 200 randomly sampled GitHub repositories and used them as data corpus. Our findings demonstrate that: (1) Approximately 3% of all issues were identified as security-related. (2) The most frequent security keywords found in the issues include: login, hash, password, inject, authentic, crypt, cookie, credential and certificate, among which "login" was dominantly used. (3) We identified 26 security-related topics across the issues that were mainly related to identity management and cryptography problems. Our qualitative analysis revealed 7 high-level themes and key points indicating the problems that developers face when implementing security features.

2. Related work

Software security focuses on developing secure software by ensuring security through design, proper testing, and educating developers and users on security techniques [5]. It differs from application security that protects software after development. It is discussed that operating a secured network is easier and more cost effective when running self-protecting software (i.e., properly designed and tested from a security perspective) [5]. Initiatives (e.g., [3], [4]) are taken by researchers to systematically organize security knowledge forming the foundation for software security. For example, Braun and McGraw [3] suggest three categories of security knowledge including perspective (e.g., rules and guidelines), diagnostic (e.g., vulnerabilities) and historical (e.g., historical risks), and they discuss their applications through the software development lifecycle.

The popularity of open-source software and the availability of big open data, motivated several researchers to use this data for software security purposes. For example, researchers commonly use open-source applications for evaluating static analysis tools that examine program source code for vulnerabilities (e.g., [6]). Open source applications have been also analyzed for relating architectural tactics to common vulnerabilities [7] and recommending tactics for security [8].

Pletea et al. [9] present sentiment analysis of security-related discussions on GitHub. Having analyzed 90 GitHub repositories, the authors conclude that 10% of the discussions were security-related, and they also involved negative emotions. We build on this work and use the proposed security keywords for exploring security issues on GitHub. However, our study differs from [9] in several aspects: a) we use a different dataset including 200 randomly sampled GitHub repositories, b) we analyze issues rather than comments on commits/pull requests, c) we particularly investigate security themes and topics emerged from posted issues.

3. Research method

This section describes the research methods used for conducting our study. We elaborate on the characteristics of our data corpus and analysis methods. We used topic modeling as well as qualitative analysis. Topic modeling was applied to the whole data corpus to extract key security topics at an abstract level. We complemented the results of topic modeling through qualitative analysis of a subset of security issues.

3.1. Dataset

We used GitHub as the main source of data. Using a crawling application, we collected issues from 200 randomly selected GitHub repositories[1]. All the repositories had at least 500 commits and at least 100 pull requests or issues. This strategy was taken to ensure that the data could provide an overview of the prevalence of security issues on GitHub, and only active GitHub projects were part of the sample. The most common programming languages in our sample were the same as in all of GitHub[2]. We organized all the issues in a single CSV (comma-separated values) file. For each issue, we included related information such as: name of the repository, title and body of the issue, timestamp of posting the issue and the issue status (open/closed). In addition, each issue was examined to determine whether it was security related. For this purpose, we used the set of security keywords provided by [9]. We marked an issue as security-related, if it (title or body) contained any of the security keywords. We initially ran a pilot study and manually verified the relevancy of 50 issues, which were automatically marked as security related. This phase helped us to ensure all the required constraints (e.g., when looking for 3-character terms) are in place as suggested by [9], and decrease the rate of false positives.

3.2. Topic modeling

Topic modeling is a Natural Language Processing (NLP) technique to automatically extract topics out of a corpus of textual data [10]. A topic refers to a collection of words that frequently co-occurred in the analyzed documents, and they are often semantically interrelated [10], [11]. Analyzing a data corpus through topic modeling enables a researcher to organize data in the form of semantic structure without pre-assumed knowledge about the content [10]. Latent Dirichlet Allocation (LDA) [11] is a popular topic modeling technique, which is commonly used by researchers for mining software repositories [10]. The LDA model provides flexibility in topic modeling by providing possibilities to treat: a) a document as a member of multiple topics and, b) a topic as a mixture of words that could belong to multiple topics [11]. Applying topic models, a data corpus could be categorized into a different number of topics (i.e., called k-value), varying from a few (e.g., k=4) to several (e.g., k=100) topics. Yet, it is a researcher's job to choose an appropriate k-value that can represent the number of

[1] Selected Repos listed at: http://tinyurl.com/SecurityIssuesProjects.
[2] https://octoverse.github.com

key topics of a corpus. We decided to use topic models based on the LDA technique due to its flexibility and proven suitability in mining software repositories [10]. We performed our analysis using well-known R libraries that support text mining, particularly the "topicmodels" [12], "tm" [13], [14] and "ldatuning" packages, and consulting with related materials provided by [15]. Topic modeling was applied to the security issues retrieved from all 200 repositories. To do so, we chose the issues from our dataset, which were marked as security-related (i.e., containing any of the security keywords).

3.2.1. Pre-processing steps. Before applying topic modeling, it is required to clean the data through pre-processing steps. We performed the following activities:

- Removing code snippets from the content of issues using regular expressions.
- Removing irrelevant words using "stop-words": We used the existing list of stop-words provided in the *"topicmodels"* R package [12]. This list includes 1000+ common English words designed for analyzing natural language texts. Running topic modeling and examining the results, we added a few more words (e.g., the names of repositories) to the list of stop-words.
- Removing punctuation, numbers and extra white space between terms.
- Transforming all characters to lower-case.
- The words were transformed into their root. Stemming was used to avoid formation of topics with various forms of a term (e.g., inject, injection, injects, injected, and injecting).
- Forming documents: In our study, we expected the LDA algorithm to treat the issues of each repository as a document. This choice was made since treating each issue as a separate document would have resulted in too many documents with almost no content. Thus, all security issues (title and body) of each repository were merged together to form a single row in the pre-processed data corpus.

3.2.2. Applying topic modeling. We ran the LDA algorithm on the pre-processed data using the "LDA()" function from the "topicmodels" package. In order to tune the input parameter (i.e., number of topics) of the LDA model we used the "ldatuning" R package. This package facilitates selecting an appropriate k-value for a given dataset. It implements and compares the results of four different metrics to find the optimal number of topics. The implemented metrics are based on maximization [16], [17] and minimization [18], [19] approaches. A researcher could analyze the extreme values of these metrics in order to determine the best value for the number of topics. For a detailed discussion on implementation and use of these metrics, we refer readers to the related references. We have examined these metrics for k=5 to k=30. The results suggested the best number of topics to be between k=17 and k=26. We chose k=26 for our analysis to ensure that the entire range of key topics are included. Besides, three metrics reached a reasonable value for k=26, indicating its appropriateness.

3.3. Qualitative analysis

We qualitatively analyzed a subset of security issues retrieved from GitHub repositories. This round of analysis was performed to a) verify issues marked as security-related, b) investigate the main themes of security issues through manual analysis and an in-depth interpretation. We selected the top 10 repositories with the highest rate of security issues and qualitatively analyzed 234 issues. We performed thematic analysis [20] on the content of all security issues from the selected repositories. The security issues were imported into Nvivo (i.e., a qualitative analysis tool) and manually open coded. In Nvivo, the issues were organized in form of datasets, i.e. one dataset per repository and one row in the dataset per issue. This data structure facilitated tracking distribution of themes within the repositories and issues, and ensuring all the issues being coded. The emerging codes were further categorized into themes.

4. Findings

We present our findings to answer the research questions of the study.

4.1. Distribution of security issues

We present the general findings from the exploration of security issues. Table 1 highlights the overview of our dataset. Our dataset included 64,963 issues posted from Feb 2007 to Aug 2016. Examining the issues against the list of security keywords [9], we have identified 1,938 security issues (i.e., ~ 3% of all the issues).

Table 1 - Summary of dataset

# Issues	Dates	# Security Issues	% Security Issues
64,963	2007/02 - 2016/08	1,938	2.98%

We explored the most frequently found security keywords. Figure 1 depicts the word cloud of the identified security keywords. We observe that "login"

Figure 1 - Word cloud of security keywords

is the most dominant keyword used in the security issues, followed by "hash", "password" and "inject".

4.2. Security topics

Table 2 shows the 26 topics identified by our topic model analysis. For each topic, we have provided the top terms of the topic. We have manually assigned descriptive labels to each topic based on a combination of terms and investigation of the data whenever it was required. Topic model analysis suggested several topics regarding identity management that contained words such as login, password, authent and credent among their top-terms. These topics covered a broad domain incorporating different authentication/authorization mechanisms (e.g., web authentication (T2), windows authentication (T13), HTTP cookie authentication (T8), certificate authentication (T20), user authentication (T5), managing user accounts/use of OAuth (T6)), as well as feature implementation and configuration supporting user credentials (e.g., login implementation in websites (T12), password UI features (T9), password provisioning and verifications to server (T17)).

We found several topics related to the concept of cryptography and encryption. These topics included areas such as: crypto-currency mining (T1), developing hash storage functionalities (T3), cryptography algorithms (T4), public-key signature system (T25) as well as encryption of sensitive data (T15 and T23). The topic models suggested 3 topics about injection (i.e., T14, T18 and T19). Having investigated the contents of these topics, we found that they were not related to security (e.g., SQL injection), but about a particular design pattern (i.e., dependency injection). There were topics from other domains such as media player security (T10), game development (T16), certificate management (T21) and violation management (T22). The combination of top terms appearing in some of the topics (i.e., T7, T11, T24, and T26) was too varied and did not clearly reflect a concept. While we have given labels to these topics based on some of the top terms, there is a possibility that they contain other contents.

Table 2 - List of identified topics and top terms

Id	Topic Label	Top Terms	Explanation
T1	Crypto-currency mining	hashrat, pool, login, password, user, hash, account, set, network, rate	Refers to crypto-currency mining and possibly managing users (login, password, account) in this context.
T2	Web/LDAP Authentication	login, password, ldap, user, web, authent, server, connect, interface, log	Refers to web authentication and possibly relates to the use of LDAP for this purpose.
T3	Hash Storage Implementation	sign, key, hashdict, set, log, modul, hashset, creat, compil, iex	"hashdict", "hashset" refer to Java classes for storing and retrieving data using hash tables. Other terms also relate to implementation.
T4	Cryptography Algorithms	checksum, sha, line, file, encod, hash, download, build, time, packag	Refers to cryptography algorithms (e.g., sha, hash). Other terms are more about implementation.
T5	User Authentication (sessions/cookies)	login, password, user, cooki, authent, session, server, connect, set, secur	Refers to user authentication and possibly use of cookies and sessions in this regard.
T6	Managing User Accounts (use of OAuth)	Login, password, user, sign, page, server, account, twitter, set, log, oauth, facebook, ssl	Refers to managing user accounts (e.g., user, account, login) and possibly the use of the OAuth protocol in this regard (e.g., twitter, oauth, facebook, ssl).
T7	Data Sanitization	Function, key, origin, report, check, sanit, codegoogl, secur, version, cba, sign, unsign, regexp	Combination of terms does not clearly reflect a topic. It is partly about data sanitization (e.g., sanit, key, origin, sign, unsign, regexp).
T8	HTTP Cookie Authentication	User, cooki, certif, tenant, sign, authent, hash, server, header,	It is about HTTP cookie authentication (e.g., cooki, authent, hash, header, request, respons).

		request, respons	
T9	Password UI Features	Password, login, user, sign, page, email, link, chang, box, send	Refers to passwords and supporting UI features (e.g., chang, box, page). For example, change forgotten password.
T10	Media Player Security	Track, video, add, secur, test, hack, url, updat, locat, herm, android	Refers to security (e.g., secur, hack, test) aspects in context of media player (e.g., track, video).
T11	Server Security Configuration	link, server, login, add, secur, Ubuntu, tile, compil, access, trusti, instal, fail, test, port, instanc, integr	Does not clearly reflect a topic. Partly refers to server security configuration (e.g., server, login, access), likely about integration servers (integr).
T12	Login Implementation in Websites	Login, site, user, jetpack, page, form, email, sso, comment, option, component	Refers to login and its implementation in websites (e.g., site, jetpack, form). Also, contains SSO (i.e., a shared authentication technique).
T13	Windows Authentication	User, authent, password, window, server, header, run, virtuoso, code, client, explor, internet, ntlm, kerbero	Refers to user authentication in Windows-based systems (e.g., window, internet, explor, ntlm, kerbero).
T14	Dependency Injection Implementation - 1	Inject, compon, servic, href, chang, test, creat, code, call, url	Refers to the dependency injection design pattern. Not related to security, as no related word found in top terms.
T15	Cookie Value Encryption	Cooki, decod, encod, data, header, yield, string, support, request, handl	Refers to cookies and encrypting their values (e.g., decod, encod).
T16	Game Development Security Configuration	Uniti, extra, render, declar, function, length, forward, password, line, add, signatur	Refers to development (e.g., function, line) in the context of games (e.g., uniti[3], render) and covers security-related aspects (e.g., password, signatur).
T17	Password Provisioning and Verification	Password, inject, user, connect, box, vagrant, fail, run, fix, access, server, framework, test, violat,	Refers to providing and verifying passwords to servers (e.g., password, access, server, violat). E.g., verifying password to connect to server.
T18	Dependency Injection Implementation - 2	Inject, compon, servic, injector, test, depend, provid, constructor, browser, router	Refers to the dependency injection design pattern. Not related to security, as not related word found in top terms.
T19	Hierarchical Dependency Injection Support	Inject, servic, parent, injector, class, child, chang, direct, depend, provid	Refers to the dependency injection design pattern. Not relates to security, as no related word found in top terms.
T20	Certificate Authentication	Certif, authent, github, server, configur, check, oauth, set, log, ssl	Refers to certificate authentication (e.g., certif, authent, oauth, ssl), which is used to secure client-server network connection.
T21	Certificate Management	Certif, sign, set, credenti, fail, profil, cert, run, platform	Refers to managing certificates (e.g., certif, install, run, fail). Could also relate to certificate profiles, i.e., used for certificate configuration.
T22	Violation Management	Violat, tabl, file, page, filter, oauth, account, secur, type, url	Refers to violation management, possibly through UI features (e.g., tabl, filter, page).
T23	User Credentials (Encryption)	Login, password, user, encrypt, page, data, system, server, button, screen	Refers to managing user credentials, through encryption (e.g., encrypt) and UI (e.g., page, scree, button).
T24	Git Configuration/ Spam Issue	Password, git, login, authent, wallet, implement, support, spam, doge, block, salt.	Does not clearly reflect a topic. Partly refers to development issues using Git, e.g., *git-salt*. Also, it contains issues about spamming in the context of crypto-currency mining (e.g., wallet, doge, spam).
T25	Public-key Signature System	Key, transact, sign, add, signatur, credenti, support, creat, code, api	Refers to a public-key signature system (key, transact, signatur) and supporting it in programs (e.g., add, support, api).
T26	Cookie (nonce authentication)	Form, cooki, page, check, fail, email, nonc, secur, submit, plugin	Does not clearly reflect a topic. Partly refers to cookie nonce authentication (e.g., cooki, nonce, secur, plugin).

[3] Unity (i.e., stemmed to uniti) is a popular game engine.

5. Qualitative results

Topic modeling treats documents as a bag of words without understanding the semantic meaning of the text. Therefore, we qualitatively analyzed a sample of security issues (i.e., from the top 10 repositories with the highest rate of security issues) for in-depth understanding. Table 3 demonstrates our manual and qualitative verification of the main topics that were identified using topic modeling. By main topics we mean the topics that were assigned to these repositories with highest probability. Out of 10 repositories, we verified the main topics of 6 repositories as relevant, 3 repositories partially relevant, and 1 repository irrelevant. We observed that the identification of main topics by LDA was more accurate for the repositories that a) were specialized in a particular domain (e.g., cryptography library), b) had larger number of issues (i.e., larger data corpus). We found that the allocation of T7 (i.e., Data Sanitization) to the contents of a repository was irrelevant.

Table 3 - Top 10 repos with high rate of security issues – allocated topics vs. manual verification

Repo	% Sec. Issues	Type/Domain of Repository	Main Topic	Manual Verification relevant: ✓ partially relevant:❖ irrelevant: ✗
crypto-js	39.74%	Library of cryptography algorithms in JavaScript	T4	✓ Majority of issues were about cryptography, stating problems (e.g., wrong hash/cipher).
yourturn	20.69%	Frontend console of STUPS platform	T22	✓ Majority of issues were about visualization of credential violations (e.g., tabular format, filtering options).
waffle	20.41%	Windows Authentication Framework	T13	✓ Topic relates to type of the repository. Majority of issues were about handling protocols (e.g., Kerberos, NTLM).
evenHire	15.38%	Web application supporting hiring workflow	T7	✗ Not related to topic. Most of the issues were about UI-related problems of the login page (e.g., need of password verification text box).
sigh	13.64%	Application for automatically handling Provisioning Profiles (PP)	T21	✓ Topic relates to type of the repository. Majority of issues were about setting up PPs (e.g., find/install certificates for PP setup).
formio	13.64%	Form and API engine for building server-less data management applications	T5	❖ Topic is partially related. Most of the issues were general authentication/authorization issues (e.g., session expiration, not getting authorized at server when giving credentials)
httparty	11.74%	Ruby library supporting the implementation of web APIs and HTTP authentication	T8	✓ Topic is highly related. Majority of issues were about authentication varying from implementation (e.g., handling authentication headers) to vulnerabilities (e.g., improper management of cookie values).
communityshare	11.54%	Web application	T2	❖ Partially related. Majority of issues were about web authentication, but mainly UI related (e.g., change password).
webob	11.45%	Python library that facilitates HTTP authentication implementation	T15	✓ Topic is related. Majority of issues were about problems in parsing cookie values in HTTP header (e.g., issue when ‘ “ ‘ is used). Yet, not much about encryption.
anahita	11.11%	Social networking platform	T13	❖ Topic is partially related. The issues were about handling authentication headers between client/server. Yet, not about Windows.

Besides verification of the topics, we identified the key themes of security issues. Table 4 summarizes the qualitative findings. Our analysis revealed that the majority of security issues were about the problems that developers face when implementing security features (i.e., 201 out of 234). The remaining issues were either not security-related or not understandable. We did not find any issue reporting critical security threats or vulnerabilities (e.g., XSS and SQL injection). As it can be seen in Table 4, we have classified the

issues about implementing security features under high-level categories including: 1) authentication/authorization issues, (2) cryptography issues, (3) handling security protocols/standards, (4) UI features, (5) error-exception handling, (6) managing cookie values/HTTP headers, and (7) others.

Our analysis revealed 29 issues related to authentication/authorization concepts with the key themes that are specified in the table. Most of the issues in this category were related to implementing shared authentication/authorization mechanisms (i.e., OAuth and SSO). For example, there were issues indicating OAuth configuration problems, need of managing flow of OAuth 2.0 tokens in the application and SSO failure in the applications. The other common issues in this category were about handling authentication headers (e.g., parsing/encoding bugs of authentication headers), issues about digest authentication (e.g., failure/need of support), and email authentication problems (e.g., error in sending password to users via email). We found a few issues indicating improper implementation of authentication/ authorization mechanisms. They include: application bugs in showing contents without authentication, wrong access control verification, and enabling deleted/deactivated users to reset password.

Under the category of cryptography, we found 58 issues, all belonging to one repository (i.e., crypto-js). This repository is a JavaScript library implementing different cryptography algorithms including hashers (e.g., MD5) and ciphers (e.g., AES). Most of the issues in this category were related to incompatibility of the cryptography solutions with different OSs (e.g., iOS, Blackberry), programming languages (e.g., C#, PHP) and applications (e.g., Browsers, PDF readers). In addition, wrong cryptographic output (e.g., hash, cipher) and lack of supporting different data types (e.g., byte, string, hexadecimal) were frequently reported. The rest of the issues indicated performance problems, suggestions to improve crypto solutions (e.g., adding pubic-key cryptography) and some general implementation questions.

We found 17 issues about handling security protocols and standards. These included issues about SSL violations (e.g. certificate verification failure on Windows using httparty), Kerberos (e.g., small size of Kerberos token in waffle), NTLM (e.g., error in transmitting NTLM token), SPNEGO (e.g., browser error in transmitting SPNEGO authentication data) and the Spring framework (e.g., error in using the framework in waffle).

Our analysis revealed 49 issues related to User Interface (UI) features with two main themes: managing credentials (e.g., login, password) and credential violations.

Table 4 – Issues in developing Sec. features

	Theme (frequency)	#
Authentication/Authorization Issues	• Shared Authentication/Authorization Mechanisms (OAuth, SSO) (12) • Handling Authentication Headers (4) • Issues with Email Authentication (sending Passwords) (4) • Digest Authentication Issues (3) • General Inquiries (1) • Possibility to reset password for deleted/deactivated accounts (1) • Showing Content without authentication (2) • Access control for service provider (1) • Putting user in incorrect group with wrong access control (1)	29
Cryptography Issues	• Incompatibility of Cryptographic solutions with different platforms (18) • Wrong Cryptographic output (e.g. hash, cipher) (18) • Limitation of Cryptographic solutions with data types (11) • Additional features to Strengthen Cryptographic Solution (5) • Performance issues with Cryptographic solutions (4) • Seeking input to Implement Cryptographic Solutions (general enquiries) (2)	58
Handling Protocols	• SSL Violation issues (6) • Kerberos (4) • NTLM (3) • SPNEGO (3) • Spring Security Framework (1)	17
UI Features	• Managing and Representing Credential Violations (37) • Managing Credential (e.g. login, password) setup (12)	49
Exception Handling	• Exception in Checking invalid credentials (2) • Lack of Exception Handling when dealing with malformed authorization header (1)	3
Cookies/HTTP Headers	• Issues of Managing Cookies (13) • Over-writing default HTTP header values (3) • Ignoring HTTP cookie values causes manipulation of load balancer (1) • Mutating cookie-hash in string (1) • Over-writing hash header content (1)	19
Others	• Setting up Provisioning Profiles (24) • Logging and Showing Access Controls (2) • Configuring Permissions (1)	27
		Total = 201

We found several issues about credential setups varying from suggestions (e.g., adding login feature, reset password, waiting message when loading authentication data) to errors (e.g., poorly rendered login page with lower resolution/small browser page size). The issues about managing credential violations were from one repository (i.e., yourturn), which provides a front-end console for STUPS[4]. Given the type and domain of the repository, it contained several issues about managing and representing credential violations (e.g., demonstrating a list of violations in tabular format with different filtering options).

We found 19 issues with regards to handling cookie values/HTTP headers. Among them several issues were about cookie management varying from handling HTTP cookies (e.g., error in setting cookie expiration time, error in parsing cookie hash values, error in sending cookie value) to same-site cookies (e.g., suggestion to support these cookies for preventing XSS attacks). Furthermore, we identified a few issues in this category indicating implementation faults that could expose applications to security threats. For instance, we observed bugs that cause over-writing default values of cookies passed in HTTP headers. These values are important as they are used to identify the server to which the HTTP request should be sent. Hence, a wrong value could redirect the request to an unwanted server or cause unexpected errors. Similarly, there was an issue indicating problems in transmitting HTTP cookie values that were set and used by a load balancer. Some of the load balancers operate with cookie-based sessions to ensure an HTTP response gets back to the same node that sent a request. Hence, errors in maintaining and transmitting these cookies could enable an attacker to interfere in the load balancer function and manipulate sessions.

Unexpected errors and failure to handle exceptions are counted among the key areas of software security problems [21] that could expose a system to security risks [22]. We have identified 3 issues related to faults in error-exception handling. For example, it was required to provide proper exception handling in the system when dealing with a malformed authorization header. Or, it was necessary to control unexpected errors thrown by an authentication framework (i.e., waffle) when verifying invalid credentials.

The rest of the issues were classified as others. This category included several issues related to setting up Provisioning Profiles (PP), i.e., used to uniquely assign a device to an iOS developer for testing apps. These issues belonged to one repository (i.e., sigh) that automatically configures PPs and included errors (e.g., in finding and installing certificates for PPs). In addition, suggestions for logging and representing audit information, and general inquiries were included in this category.

6. Limitations

We realize that our study is limited to a particular population of software repositories that are hosted on GitHub and are active (considering number of commits, pull requests and issues). Our data provides some degree of representativeness due to our attempt to not filter repositories based on attributes such as type or business domain, yet it does not comprehensively include all open source software repositories. Furthermore, there are limitations associated with the applied analysis methods. We identified security issues using a list of security keywords proposed by [9]. This approach is associated with the risk of including issues that contain keywords but are not related to security. We tried to address this risk and minimize the rate of false positives by performing a pilot study and manually verifying selected issues. Whilst this strategy significantly helped us to improve our search and identify relevant security issues, there is a chance of having false positives in the analyzed data. An example of this situation is given by issues containing “inject”, which were not related to security (e.g., SQL injection) but referring to a design pattern (i.e., dependency injection). In addition, topic modeling has proved to be applicable for analyzing a large corpus of textual data, yet it does not always generate sets of terms that are associated with analytically determined topics. Hence, the assigned topic labels are largely based on researchers’ opinions and associated with the risk of misinterpretations. We tried to mitigate this risk by verifying the identified topics against some of the top repositories from which the topics are generated. In case of high ambiguity, we manually checked a couple of issues from related repositories. Besides, we tried to complement the abstract results of topic modeling with an in-depth view obtained through qualitatively analyzing a sample of around 230 issues. It should be noted that qualitative findings are based on analyzing a proportion of data that may not be generalizable. Lastly, our analysis relies on repository contributors who experienced and posted the issues. We did not verify the validity of issues against the code. Naturally, our results only include security-related issues that are known and have been reported.

7. Discussion and conclusion

[4] STUPS toolsets provide audit-compliant Platform-as-a-Service

Open source software introduces opportunities as well as threats when it comes to system security [23], [24]. Availability of the source code helps attackers to manipulate software for malicious purposes [23], [24]. Conversely, users have more opportunity to increase security of open source software by applying different techniques (e.g., using auditing tools to automatically identify vulnerabilities in the code) [23], [24]. Several security researchers have investigated open source software repositories for different purposes such as evaluating effectiveness of static analysis tools in finding buffer-overflow vulnerabilities [25], understanding vulnerabilities related to architectural tactics [7] and mining emotions around security issues [9]. Many of these studies perform code-based analysis of open source software for vulnerabilities and security tactics. Taking a different approach, in this paper, we have analyzed a random sample of open source repositories for security issues. We aimed at understanding the extent to which security issues are posted in open source software repositories, and identifying their main topics/themes. Choosing a random set of repositories enabled us to investigate the prevalence of security-related issues on GitHub. We believe that our automatic search could extract any type of security issue (e.g., threat, mechanism, and standard) posted on the selected repositories, as we used a comprehensive list of security keywords for the search.

The findings from our study are two-fold: methodological learning and content analysis results.

Using topic modeling along with qualitative analysis enabled us to explore a large corpus of data at an abstract level, while getting into depth on a relatively smaller proportion of the data. Given our experience, topic model analysis is associated with a number of challenges such as: difficulties to identify the proper number of topics, difficulties in interpretation of topics based on the allocated terms. Whilst we used a systematic solution for selecting the number of topics, we observed that some of the topics (e.g., T18, T19) had overlaps and could be merged. In addition, the combination of terms in some of the topics was quite varied, and did not clearly reflect a meaningful concept. Therefore, we assert that topic modeling is a useful analytical tool enabling researchers to quickly learn about contents of a data corpus. Yet, its results are highly abstract, and need to be complemented with other analysis methods. When we verified the topic modeling against the qualitative findings, we observed that topic modeling produced more accurate topics when data volume was larger, and more specific (e.g., cryptography). We verified most of the topics as properly assigned to the repositories, yet one topic (i.e., T7) was found irrelevant in the context of the assigned repository.

Our analysis demonstrated that the most frequent security keywords used in the issues were login, hash, password, inject, authentic, crypt, cookie, credential and certificate, among which "login" was dominantly used. Applying topic-modeling analysis on the issue contents, we found that the majority of the security issues were about identity management (authentication/authorization/credentials) as well as cryptography. Despite frequently finding the "inject" keyword and identifying related topics, our investigation revealed that these topics mainly referred to a popular design pattern (i.e., dependency injection) rather than security threat/attacks (e.g., SQL injection). Our experience in this regard could be used by other researchers who are interested in mining security issues using keywords.

Furthermore, our qualitative findings revealed that the majority of analyzed issues were problems that developers face when implementing security features. We did not find any issue reporting security vulnerability or an attack among those issues that we manually analyzed. Similarly, topic modeling did not reflect well-known vulnerability topics. We argue that this might be partly due to the development-centric nature of GitHub issues. In fact, we observed that GitHub repository owners tend to use issues for tracking bugs. They sometimes maintain different means (e.g., public mailing list) for discussing other matters. There is a chance that critical security problems are not posted as GitHub issues. A future study could explore other communication channels for open repositories for analyzing security problems. In addition, we discuss that security vulnerabilities cannot be easily encountered without running tests designed for this purpose and/or the use of related tools (e.g., vulnerability scanners). Besides, identification of (critical) security bugs requires security knowledge and expertise that might not be in skillsets of the developers who contribute to a repository. These might also be the reasons for not observing issues reporting security threats. Lastly, this observation might be due to limitations of our analysis methods. Topic modeling tends to reflect topics at an abstract level. It might not be effective to extract particular vulnerabilities. While the qualitative analysis enabled us getting into depth, practically it was not possible to manually analyze all the issues.

8. Future work

Our work can be extended from several angles in the future. In this study, we explored a sample of GitHub repositories from the security perspective.

Future studies can explore other open source repositories (e.g., on SourceForge, Bitbucket) and compare the results. Also, more samples from GitHub can be explored longitudinally for potential trends. In addition, a future study can explore correlation of security issues with characteristics of repositories (e.g., programming language, domain, number of lines of code, number of contributors). Our work can be extended by analyzing a larger pool of data for behavioral patterns (e.g., number of comments or openness) and possible differences between security and non-security issues.

The security topics and themes that emerged from this study can be further explored and structured towards areas of knowledge that developers require for implementing security features. In this regard, comparing our findings with existing security classifications (e.g., CVE) is highly considered in the future.

9. References

[1] P. Institute., "The State of Mobile Application Insecurity.," 2015.

[2] C. Weir, A. Rashid, and J. Noble, "How to improve the security skills of mobile app developers? Comparing and contrasting expert views," in *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*, 2016.

[3] S. Barnum and G. McGraw, "Knowledge for software security," *IEEE Security & Privacy,* vol. 3, pp. 74-78, 2005.

[4] A. Hazeyama, "Survey on body of knowledge regarding software security," in *Software Engineering, Artificial Intelligence, Networking and Parallel & Distributed Computing (SNPD), 2012 13th ACIS International Conference on*, 2012, pp. 536-541.

[5] G. McGraw, "Software security," *IEEE Security & Privacy,* vol. 2, pp. 80-83, 2004.

[6] V. B. Livshits and M. S. Lam, "Finding Security Vulnerabilities in Java Applications with Static Analysis," in *Usenix Security*, 2005.

[7] J. C. Santos, A. Peruma, M. Mirakhorli, M. Galstery, J. V. Vidal, and A. Sejfia, "Understanding Software Vulnerabilities Related to Architectural Security Tactics: An Empirical Investigation of Chromium, PHP and Thunderbird," in *Software Architecture (ICSA), 2017 IEEE International Conference on*, 2017, pp. 69-78.

[8] M. Mirakhorli, J. Carvalho, J. Cleland-Huang, and P. Mäder, "A domain-centric approach for recommending architectural tactics to satisfy quality concerns," in *Twin Peaks of Requirements and Architecture (TwinPeaks), 2013 3rd International Workshop on the*, 2013, pp. 1-8.

[9] D. Pletea, B. Vasilescu, and A. Serebrenik, "Security and emotion: sentiment analysis of security discussions on GitHub," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, 2014, pp. 348-351.

[10] T.-H. Chen, S. W. Thomas, and A. E. Hassan, "A survey on the use of topic models when mining software repositories," *Empirical Software Engineering,* vol. 21, pp. 1843-1919, 2016.

[11] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of machine Learning research,* vol. 3, pp. 993-1022, 2003.

[12] K. Hornik and B. Grün, "topicmodels: An R package for fitting topic models," *Journal of Statistical Software,* vol. 40, pp. 1-30, 2011.

[13] D. Meyer, K. Hornik, and I. Feinerer, "Text mining infrastructure in R," *Journal of statistical software,* vol. 25, pp. 1-54, 2008.

[14] I. Feinerer, "Introduction to the tm Package Text Mining in R," ed, 2017.

[15] J. Silge and D. Robinson, *Text Mining with R*. http://tidytextmining.com: O'Reilly, 2017.

[16] J. Cao, T. Xia, J. Li, Y. Zhang, and S. Tang, "A density-based method for adaptive LDA model selection," *Neurocomputing,* vol. 72, pp. 1775-1781, 2009.

[17] R. Arun, V. Suresh, C. Veni Madhavan, and M. Narasimha Murthy, "On finding the natural number of topics with latent dirichlet allocation: Some observations," *Advances in Knowledge Discovery and Data Mining,* pp. 391-402, 2010.

[18] T. L. Griffiths and M. Steyvers, "Finding scientific topics," *Proceedings of the National academy of Sciences,* vol. 101, pp. 5228-5235, 2004.

[19] R. Deveaud, E. SanJuan, and P. Bellot, "Accurate and effective latent concept modeling for ad hoc information retrieval," *Document numérique,* vol. 17, pp. 61-84, 2014.

[20] V. Braun and V. Clarke, "Using thematic analysis in psychology," *Qualitative Research in Psychology,* vol. 3, 2006.

[21] K. Tsipenyuk, B. Chess, and G. McGraw, "Seven pernicious kingdoms: A taxonomy of software security errors," *IEEE Security and Privacy,* 2005.

[22] M. Howard, D. LeBlanc, and J. Viega, *24 Deadly Sins of Software Security - Programming Flaws and How to Fix Them*, 2010.

[23] C. Cowan, "Software security for open-source systems," *IEEE Security & Privacy,* vol. 99, pp. 38-45, 2003.

[24] C. Payne, "On the security of open source software," *Information systems journal,* vol. 12, pp. 61-78, 2002.

[25] M. Zitser, R. Lippmann, and T. Leek, "Testing static analysis tools using exploitable buffer overflows from open source code," in *ACM SIGSOFT Software Engineering Notes*, 2004, pp. 97-106.