The Current Practices of Changing Secure Software

An Empirical Study

Ameerah Muhsinah Jamil
Iowa State University
amjamil@iastate.edu

Lotfi ben Othmane
Iowa State University
othmanel@iastate.edu

Altaz Valani
Security Compass
avalani@securitycompass.com

Moataz Abdelkhalek
Iowa State University
moataz@iastate.edu

Ayhan Tek
Cyber Electra
ayhan.tek@cyberelectra.com

ABSTRACT

Developers change the code of their software to add new features, fix bugs, or enhance its structure. Such frequent changes impact occasionally the security of the software. This paper reports a qualitative study of the practices of changing secure-software in the industry. The study involves interviews with eleven developers and security experts working on banking software, software for control systems, and software consultation companies. Through these interviews, we identified that the main security aspects are: dependency vulnerabilities, authentication and authorization, and OWASP 10 vulnerabilities. The common techniques used to assess software after code change are: code review, code analysis, testing, and keywords search. The main challenges that practitioners face are the diversity of the security issues and the lack of effectiveness of the security assurance tools in detecting vulnerabilities. The study suggests that developers of secure software need techniques that support effective security assurance of modified software.

CCS CONCEPTS

• **Security and privacy** → **Software security engineering**;

KEYWORDS

Secure code, Code change, Secure code change

ACM Reference Format:
Ameerah Muhsinah Jamil, Lotfi ben Othmane, Altaz Valani, Moataz Abdelkhalek, and Ayhan Tek. 2020. The Current Practices of Changing Secure Software: An Empirical Study. In *Proceedings of ACM SAC Conference (SAC'20).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3341105.3373922

1 INTRODUCTION

Developers change the code of a software, i.e., modify one or many parts of the software [30], to introduce new features, fix existing defects, or improve the maintainability or the performance of the software. The impact of code changes on the functionalities of the software has been extensively studied [15]. These studies focus on the impact of the change on the functionalities of the software. The common approaches rely on tests to identify defects created by code change activities. Since tests do not detect always vulnerabilities, these approaches do not apply to detect the impact of code changes on the security of software. For example, assume that we have a `writeToFile` function that uses the `FileIOPermission` constructor and assumes that a developer changes the value of `Permission State` used in the `FileIOPermisison` from `Read` to `Unrestricted`–see Listing 1. The change may be required to implement new features, but it gives the applications that can use the method the permission to indirectly write to files–similar to the set-UID vulnerability [10].

```
public void changedClass
{
   public void writeToFile(String myInput)
   {// Do Something
    FileIOPermission myPerm =
    new FileIOPermission(PermissionState.Read);
        //    Change to: Unrestricted
    myPerm.Demand();
    // Do Something
    }}
```

Listing 1: An example of risky code change.

The techniques to assess the security of software includes code review, static and dynamic code analysis, and penetration testing. There is, currently, no study that discusses how the practitioners change secure software safely, i.e., they produce secure modified-software. This paper aims to address that by answering the question: *What are the practices that developers and security experts use to ensure the security of software affected by code change?* To address this question, we interviewed eleven developers and security experts who participate in the security assessment of the software developed in their respective organizations. Then, we transcribed the interviews, extracted the main information and grouped them into themes, and analyzed the findings.
The contributions of the paper are:

(1) Identification of security aspects that practitioners look for when changing secure software.
(2) Identification of the techniques used in the industry to ensure the security mechanisms do not break after a software change.
(3) Identification of the challenges that practitioners face when changing secure-software.

The results of this work could be used by organizations when developing their secure-code change processes and by academia to develop solutions and techniques that help practitioners change secure-code efficiently.

This paper is organized as follows: Section 2 discusses related works, Section 3 describes the research approach, Section 4 presents the results of the study, Section 5 summarizes the study results and discusses the impacts and limitations of the study, and Section 6 concludes the paper.

2 RELATED WORKS

There has been extensive work on tracing code changes. For instance, Kim and Notkin [14] proposed LSdifff, a tool that infers the systematic structural difference as logic rules. LSdiff has a good ability to discover exceptions and reduce changes investigation-time but is limited to cross-language changes.

Hilton et al. [13] studied the trade-offs between speed vs accuracy, access vs security, and configuration vs simplicity. They argued that Continuous Integration (CI) improves the code validation but the thoroughness slows down the assessment of the security of changed code; CI protects the integrity of the code passing in the integration pipeline but this protection makes it sometimes difficult to troubleshoot build failures, and the flexibility in CI configuration and usage increases system complexity.

In agile development, the developers continuously change the code of the software which invalidates the security assurance of a given release of the software. Ben Othmane and Ali [18] developed secureAgile, a tool that traces the impact of code-changes on the security of software. The tool is designed as an Eclipse plug-in that visualizes the security assurance case of a given software and the mapping of the case elements to the software code. Based on the case study, the tool managed to trace the code changes to security assurance artifacts. Hence, it can be used to assess the security of the software. Abdelkhalek et al. [4] extended the work by developing a toolset that identifies the impacts of code changes on the security aspects of the previously assessed software. The toolset uses callgraph and security assurance cases techniques to model, respectively, the software code and security requirements of the software. The preliminary evaluation of the toolset using three open-source ERP/Ecommerce software applications shows the effectiveness of the approach to identify the code change impacts on the security of the software.

Vanciu [28] proposed Scoria, a semi-automatic approach to find architectural flaws, which count for 50% of security vulnerabilities. They proposed the use of annotations to mark the security-related code. The tool generates a graph representing the code and the annotations and allow to query the graph and identify potential architectural flaws.

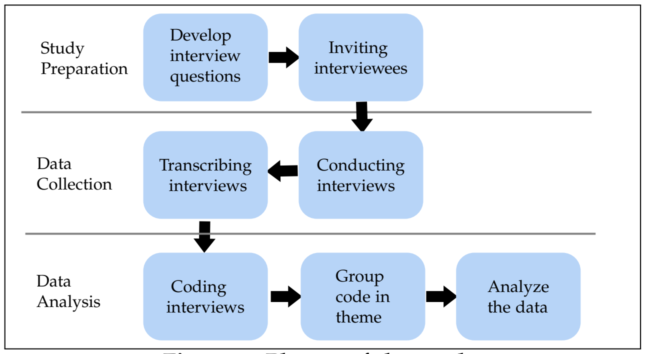


Figure 1: Phases of the study.

Mohamed et al. [17] studied the practices of secure software engineering in Malaysia. The authors found a lack of implementation of secure software practices although there is high awareness about the importance of these practices among the practitioners.

Meneely et al. [16] studied the applicability of Linus's Law [21] to security-sensitive code-changes by measuring and analyzing the participation of the developers in reviewing software. The study shows that more reviews and reviewers are associated with vulnerable files that contradict Linus Law but the developer experiences associated with the vulnerabilities follows Linus Law.

Bosu et al. [7] and Bosu [6] performed a study on the characteristics of vulnerable code changes. The study found that careful review should be done to modified files and that code changes are made often by less-experienced developers. Also, the study shows that peer-code reviews allow discovering many types of vulnerabilities and that vulnerabilities that require extra works are likely to be abandoned.

The work discussed above focuses on the improvement of the development process, the study on the practice of secure software, the tools supporting secure-code changes, and the characteristics of vulnerabilities created when changing secure-code. No study discusses how the practitioners change secure software safely, i.e., they produce secure modified-software. Answering these questions helps to design practical solutions that developers and security experts could use to be efficient in developing secure software.

3 RESEARCH APPROACH

This study aims to understand the current practices of secure-code changes in the industry. The data source of the study comes from the developers and security experts practicing secure-code change in the industry. Figure 1 depicts the process of the study, which has three phases: study preparation, data collection, and data analysis. The details of each of the phases are described in the following.

3.1 Study Preparation

The description of the study preparation follows.

Table 1: Business performed by each participant.

Participant	Business
P1	Software engineering
P2	Banking systems
P3	Security consultation
P4	Control system
P5	Control system
P6	Control system
P7	Banking systems
P8	Software engineering
P9	Security consultation
P10	Education systems
P11	Software engineering

Interview protocol. The interview protocol was constructed using Turner's guidelines [27]. First, we formulated a set of interview questions on the research goals and information from informal discussions with the project sponsor. The questionnaire was tested by trial runs with team members and revised based on the feedback. The set of question consist of six open-ended questions. The open-ended questions enabled participants to provide detailed responses.

Participants selection. We invited a set of practicing experts of secure software development from Canada and USA to the interview. Eleven experts from the banking, control systems (agriculture and avionic) and software engineering industries accepted our requests. (And we do not have any dropout case from these experts.) Table 1 shows the business of the organization of the participants. The participants requested to keep their identity anonymous.

3.2 Data Collection

The data collection consists of two sub-phases. Conducting the interviews and transcribing the interviews.

Conducting the interview. We scheduled one-hour meeting with each of the experts that accepted our invitations. The meetings were held through Zoom because the experts are located far from the interviewers. The interviews were conducted by two of the authors. At the beginning of each of the interviews, the interviewers explained the goal of the project and process to the interviewees and requested their consent to record the interview.

Transciption of the interviews. The interviews were transcribed using oTranscribe. [1]

3.3 Data Analysis

Interview coding. We used the thematic analysis method for the interview coding [23]. Interview coding uses the interview transcripts as input and outputs codes that identify all the aspects mentioned during the interviews. A code is a word or short phrase identifying the essence of a portion of language-based or visual data. At the end of this step, we assigned codes to each of the eleven interview transcripts. For example, we assigned code *OWASP top 10* to text *Mostly the, you know the OWASP top 10 threats, you know SQL injection, cross-site scripting."* We used Atlas.ti [2] tool to code the interviews.

[1] oTranscribe: https://otranscribe.com/

Table 2: Coding schema.

Code Group	Description
Software type	The type of software developed by the participant.
Security aspect & concern	The security aspect that the participant is concerned about when they change their secure-software.
Security training	The training provided to the people involved in the secure code change.
Security assessment activities	The activities performed by the organization to assess the security of the software.
Secure development process	The secure development process that the participant use.
Code change process	The activities of changing the code from request to deployment.
Techniques secure-code change	The techniques used to ensure the security of secure-code not break.
Tools	The analysis and testing tools used in the secure development process.
Process participants	People involved in secure development process and secure code change process.
Participants roles	The roles and responsibilities of the people involved in the secure development and secure-code change processes

Data Extraction and Classification. The next step was to group similar codes to form themes. A theme generalizes a set of codes belonging to a given concept as a theme. The process of assigning themes to codes was done for each transcript. Then, we merged the codes that were semantically similar across transcripts. Table 2 lists the themes and associated categories.

Analysis of the results. From the code group, we identified the information on the software type, security aspects and concerns, security training, security assessment activities, secure development process, code change process, techniques to ensure secure-code change, tools for analysis and testing, and the people involved in the process. We, then, modeled the relationships among these themes.

4 RESULTS

This section describes the main aspects related to the practices of ensuring the security of changed code, which are the types of software that the participants develop, the security aspects that their organizations consider as important, the

[2] ATLAS.ti: https://atlasti.com/

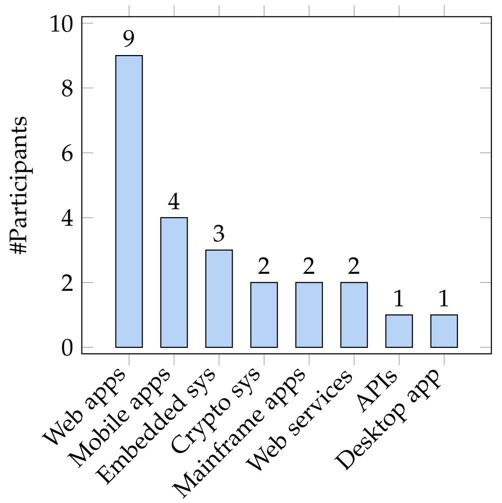


Figure 2: No. of participants develop each of the software type.

code-change process used in these organizations, the processes that they use to ensure the security of the software does not break after code changes, the tools that the participants use in the process, and the people involved along with their roles.

We used *Pi* to refer to participant *i* in the interview.

4.1 Software Type

The participants develop and ensure the security of software for banking, agriculture, avionic, education, and software engineering industries. Figure 2 shows the number of participants for each of the software types–each participant works on one or many software types. We observe that nine participants (82%) work on web applications and only one participant works on desktop applications. The reason for the focus on the security of web applications is because this software is often Internet-facing applications, which makes them target to remote attackers. For instance, the only participant (**P1**) that develop desktop application said ″You know we do have some other PC-based applications but for the most part, you know we are starting on the process of since most of our applications are developed on, in a web page application those are the one we are focusing on right now." We understand that participant **P1** do not ensure the security of desktop application because the primary focus is web applications. Also, we observe that two participants are involved in ensuring the security of legacy mainframe applications. The participants do not change the code of these applications but assess the impact of the code-changes to their software on the mainframe applications that they interact with.

Figure 3 illustrates the programming languages along with the number of participants that use them. Note that the tools used to analyze and test the security of software depend often on the used programming languages.

4.2 Security Aspects and Concerns

The types of software that organizations develop drive the security aspects and concerns that they focus on. Table 3 lists the security aspects and the number of participants that specified each of them.

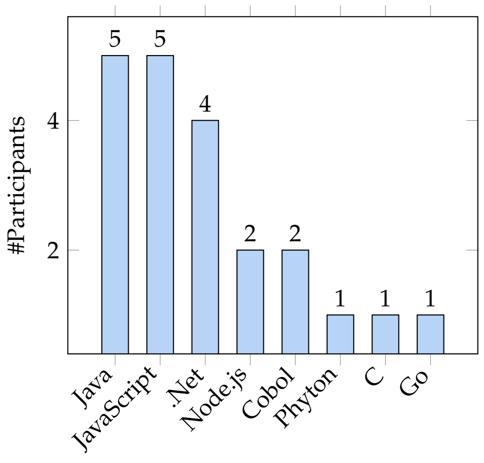


Figure 3: No. of participants using each of the programming language.

Table 3: No. of participants concerned with each of the identified security aspects.

Security aspects	# Participants
Authentication & authorization	4
OWASP top 10	4
Dependency vulnerability	3
Access control	2
Buffer overflow	1
Encryption	1

Authentication and authorizations. Correct implementation of authentication and authorization are often difficult but critical to protect the systems against e.g., credential sniffing, dictionary attack, and session hijacking attack [19]. Attackers who gain access to a given system can perform fraud, identity theft, etc. Three out of the four participants that are concerned about correct implementation of authentication and authorization about said that they do not deploy the software, either for the first release or subsequent releases before making sure that this aspect works perfectly. For instance, participant (**P7**) said *″...even before you make a change, any security control within that application... is executed when it is needed, and it has to be executed as intended″*.

OWASP top 10. OWASP top 10 [1] includes, according to the security practitioners, the most ten critical security flaws in web applications. The top ten vulnerabilities in 2017 are injection, broken authentication, sensitive data exposure, XML external entities (XXE), broken access control, security misconfiguration, cross-site scripting (XSS), insecure deserialization, using components with known vulnerabilities, and insufficient logging and monitoring [1]. Four participants stated that the OWASP top 10 are the main security concerns and that they take serious effort to ensure that their software does not have these vulnerabilities. Note that the participants specified dependency vulnerabilities and authentication and authorizations aspects separately although they are among the OWASP top 10 because they are important issues for them.

Dependency vulnerabilities. Each software is composed of a set of components (in the form of binaries, libraries, source code, or APIs [9]) that depend on each other [25]. These components could be internal or external, i.e., developed by the team or not. The external components of a given software could include vulnerabilities that impact the security of the software, known as dependency vulnerabilities. Three of the participants were concerned about the vulnerabilities in the direct and also indirect dependencies.[3]

Access Control. Access control mechanisms are used to ensure that the usage of a system complies with a set of access policies [24]. Access control vulnerabilities include bypassing access control checks, the elevation of privileges, and metadata manipulation [20]. Two participants were concerned about this aspect. For instance, participants **P10** stated: *"I focus on permissions of users: can they access, you know, certain areas in the application? Also, what sort of data are they accessing in those particular places?"*. The participant considers that the aspect is critical for learning management systems because flaws in such systems would allow tampering with the students' grades or accessing unauthorized information.

Buffer overflow. Buffer overflow occurs when the size of a destination buffer is smaller than the size of input data, which may allow overwriting the data adjacent to the buffer[3]. Attackers could exploit the vulnerability to e.g., gain control of a system or execute arbitrary code with elevated privileges. The vulnerability concerns one participant because they write software using the non-memory- safe language C.

Encryption. Encryption is used to implement several of the security features. One of the participants (**P11**) developed software that stores and manages encrypted data. The participant fears that potential attackers exploit flaws in the implementation of the mechanisms to access the secret data. Breaches in the data storage can damage the reputation of the organization.

4.3 Security Training

Four participants reported that they have/had formal training in developing secure software. Participant **P1** reported that their organization trains the developers on preventing the OWASP top 10 security vulnerabilities [1]. Advanced developers in their organization receive, in addition, security .Net training. Participant **P2** reported that their organization trains the developers on a Static Application Security Testing (SAST) tool that they use to detect vulnerabilities. Participant **P7** reported that their organization has security training tailored to each role in the organization. For instance, developers are trained on secure coding standards (defensive programming) and preventing OWASP top 10 vulnerabilities; architects are trained on threat modeling, designing the architecture for secure applications; project managers are trained on security management; and quality assurance team members are trained on verifying the security of software. Participant **P9** reported that they use a secure development process, which includes secure coding standards. The organization outsources its software development. Similar to their developers, contractor teams receive training on the development process and rules to ensure they deliver secure software.

[3]Indirect dependencies are dependencies of either direct dependencies or indirect dependencies.

4.4 Security Assessment Activities

The participants reported six security analysis activities: static code analysis, dynamic code analysis, object code analysis, software composition analysis, penetration testing, and code review.

Static Application Security Testing (SAST) activity, i.e, analyze the source code to identify vulnerabilities, is performed by nine out of the eleven participants. According to participant **P2**, the activity is performed in the development phase (every developer runs the SAST tool on their code when they wish) which allows them to identify vulnerabilities and fix them in the early stage of the development. The participant expects that the cost of fixing vulnerabilities in the early stages is much smaller than fixing them in later stages of the software life cycle phases.

Dynamic Application Security Testing (DAST) activity, i.e., execute the code and assess the software for security vulnerabilities [12] is performed by five participants.

Object code analysis is a structural coverage analysis performed on the object code. Only participant **P5** reported performing the activity. They stated: *"We do object code analysis on level A stuff"*. Which followed the DO-178C's requirements[2] for object code verification. Participant **P5** applied this analysis to inspect the output of the code.

Using components with known vulnerabilities is among the OWASP top 10 vulnerabilities in 2017. Software composition analysis activity is used to assess the exposure of a given software to known vulnerabilities by identifying the components of the software along with their licenses and related known security vulnerabilities [22]. Four of the eleven participants reported that they perform the analysis to ensure they do not deploy software that includes vulnerable third-party components.

Penetration testing [11] is a manual activity for identifying vulnerabilities in software and exploiting them. The activity is performed by eight participants. Two participants (**P4**, **P6**) do penetration testing for security-critical applications. Participant **P2** performs full tests on the code-base of the first release of a given application (e.g., ensuring file permission is correct) and assess the risks associated with each subsequent code-changes before deciding on the tests to perform on the application.

Code review is reading the code line by line to find flaws and check its compliance with the coding standards. The activity is performed by four participants. Participant **P7**

Table 4: No. of participants performed each type of security assessment.

Security assessment	# Participants
Penetration testing	8
Static analysis	8
Dynamic analysis	5
Code review	4
Software composition analysis	4
Object code analysis	1

performs code review from the beginning throughout the development process while participant **P6** does peer review at the end of each release of the given software.

4.5 Secure Development Process

Developing secure software requires the integration of security activities into the development process [26]. The interview participants reported that they use a set of security policies and standards to produce secure software. These include secure coding standards, application security assessment standards, security testing standards, and rules for deploying applications–e.g., software are not allowed to be deployed to the production environment if they include known critical vulnerabilities.

Some of the interviewed participants described the processes that they use to ensure the security of their modified software. For instance, Figure 4 depicts the vulnerability fixing process used by Participant **P2** organization, a bank. The organization uses a static and dynamic code analysis tools to check, on-demand, their application code for security vulnerabilities. The developers fix the vulnerabilities that are familiar with and request assistance from the security team to fix the rest of the identified vulnerabilities. Participant **P7** works also in banking but their organization uses another process. They reported that their development process incorporates security activities in all the development life-cycle phases: they perform a preliminary software risk analysis and elicit the security requirements in the requirements elicitation phase; perform security architecture, threat modeling, and security test planning in the design phase; perform code review and analyze the code using DAST and SAST tools to identify language-level vulnerabilities in the development phase; perform security testing including penetration testing in testing phase; ensure secure deployment and verify the configuration of the environment in the deployment phase.

Participant **P4** reported that their team has secure by design process that integrates secure software development practices. The team evaluates the criticality of each modified application and assesses the business impacts of potential compromises of the given application. They evaluate the criticality of these applications and selects the appropriate security activities (code analysis, architectural design review, vulnerability test, and penetration testing) accordingly.

Participant **P9** reported that their organization incorporates three security activities into their software development process, which are: elicit the security requirements of the given application and design solutions that address them; perform source-code analysis and testing to check the code for vulnerabilities; and monitor the deployed applications to identify potential attacks.

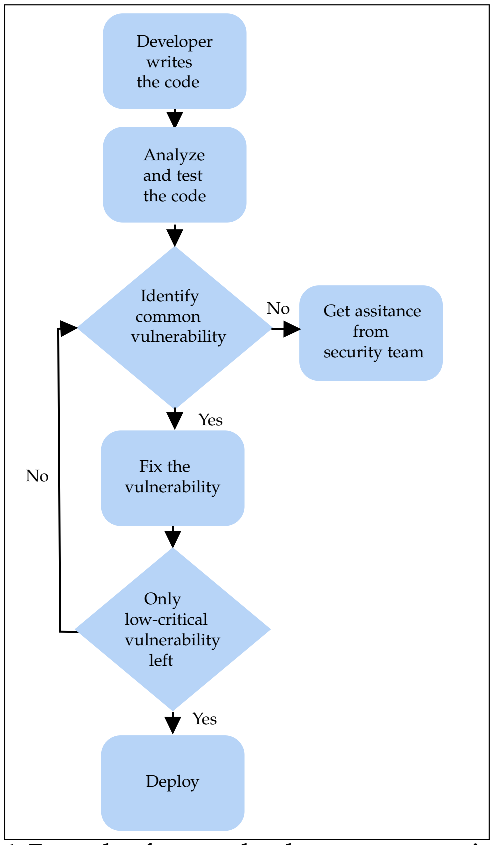

Developer writes the code
Analyze and test the code
Identify common vulnerability
No
Get assitance from security team
Yes
Fix the vulnerability
No
Only low-critical vulnerability left
Yes
Deploy

Figure 4: Example of secure development process in participant P2 organization.

Participant **P6** works on avionic control systems. The team does not include security activities in their development process because the systems that they develop need to comply with a well-specified set of requirements and are supposed to work in closed network systems. The team, however, works on including secure-coding standards, security requirements, language-based vulnerabilities testing, and penetration testing for their software into their development processes.

4.6 Code Change Sub-process

Changing the code of deployed software is commonly performed by nine participants. The remaining two participants (**P5** and **P6**) do not frequently change their software. These participants work on safety-critical systems, such as flight control systems. The participants usually have their software certified by a third-party before having them deployed. Changing this software is costly and risky. It implies often rewriting the software from scratch, with different ideas and constraints.

The interviewees reported that they change their code to enhance their software, address small change requests,

Table 5: No. of participants performed each technique to ensure secure-code change.

Techniques	# Participants
Review	8
Code analysis	5
Testing	5
Keywords search	1

refactor the code, or address new business or legal policies. These changes could be ad-hoc or periodic, i.e., once a week.

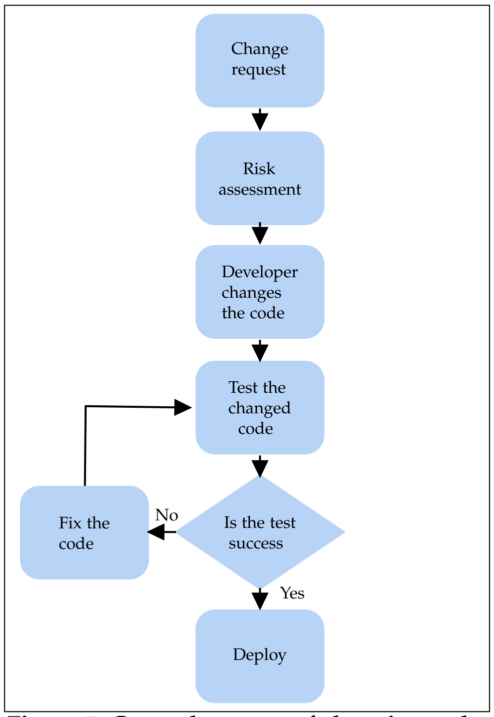


Figure 5: General process of changing code.

Figure 5 depicts the generic process of code change used by the organizations of six of the participants. The process starts with a change request followed by an assessment of the security risks. The developers, then, change the software and test it. They make the software available for deployment if the tests are successful and fix the code and iterate on the tests otherwise. The participants apply the process differently. For instance, the organization of participant **P7** has a change control board that reviews the changed code, evaluates the rationale for changing the code, checks the test results, and verifies if the changed code introduces new vulnerabilities; and three participants (**P3**, **P7**, **P9**) have post-production operation where they monitor the code changes after deployment.

4.7 Techniques to Ensure Secure-code Change

The participants use four techniques to ensure secure-code change: (1) review, (2) code analysis, (3) testing, and (4) Keywords search. Discussions on each of these techniques follow.

Review. Eight participants reported that they perform code review to ensure that the changed code does not break the existing security mechanisms that the software implements. Three out of the eight participants review only the changed pieces of code because they find it impossible to review the code-base of their large applications regularly. In addition, two of the eight participants perform security manual-review and two participants perform security peer-review. Participant **P9** stated that having checklists of do and don't help perform the code review. They stated that although code reviews are good in ensuring that the code is written according to the standards, and the code did not introduce new vulnerabilities they are labor-intensive, hard to do, very expensive, and take a long time to complete.

Code Analysis. Five participants reported that they perform SAST and/or DAST on the software that they change. Some tools that the participants use in this assessment activity are: Fortify, CodeSonar, and Lint.

Testing. Five participants reported that they perform functional and penetration testing on the modified software. Among the five participants, participant **P9** uses a test-driven development method. Therefore, they perform all the unit tests and integration tests on the modified code. They perform, in addition, a penetration testing on the changed code. Participant **P2** assesses the risks of their code-changes and have the project manager decides on whether to perform penetration tests on the changed code-base or not.

Participant **P2** suggested preparing a checklist that includes a list of criteria to verify the quality of the penetration testing, which would include the security features that should be tested, e.g., file permission and authorization. The risk manager verifies if the results of penetration testing of a given software comply with the checklist and provides approval for deployment to the production when appropriate.

keyword search. Participant **P11** reported that they use *keywords* to locate code changes that require code reviews by security experts. The participant specifies the keywords that need to be checked in the configuration file of their tool. The tool scans the source code files of the given software and uses the keywords to identify security-relevant code changes. The developers could add a special annotation (considered as keywords) to their code chunks to alert security experts to review the changed code.

4.8 Tools for Analysis and Testing

Table 6 lists the security tools that are used by the participants along with the number of participants that use each of them.

The participants reported that choosing the perfect tools is difficult because each tool has specific purpose and set of functionalities, and addresses a specific set of vulnerabilities. Several factors need to be considered in choosing the tools including the programming languages, the used frameworks, the required level of experience of the user, and the age of the application. For instance, participant **P9** states: *"There's a lot of different languages right! Especially, I have seen a lot of shops, each shop supports different languages, environments, [...] frameworks, required level of experience, and age of the applications. [...] [There*

Table 6: Tools and usage.

Tools	Usage	# Participants
Veracode	Static analysis	4
Fortify	Static analysis	3
SonarQube	Static analysis	3
Coverity	Static analysis	2
CodeSonar	Static analysis	1
FindBugs	Static analysis	1
Lint	Static analysis	1
Splint	Static anaysis	1
ToolPAL	Static analysis	1
Checkmarx	Static analysis, security testing	1
Sonatype	Component analysis	2
Dependency Check	Vulnerability analysis	1
AppScan	Web app security testing	1
Arachni	Web app security testing	1
WebInspect	Web app security testing	1
Jest	JavaScript testing	1
Burp Suite	Penetration testing	3
Keyword tool	Vulnerability testing	1
SD Element	Understand requirements	1

are] different security tools that work the best for each of these." The participant suggested having a score sheet that helps to select the appropriate tools for the given task based on their support of the factors that they cited. The scores should be given based on an objective evaluation and should include ease of use, ease of deployment, budget, etc.

4.9 Participants Roles

Table 7 lists the different roles that contribute to producing secure-code change that was reported by the participants.

4.10 Challenges

The participants reported three major challenges that they encounter when developing secure software. The details about these challenges follow.

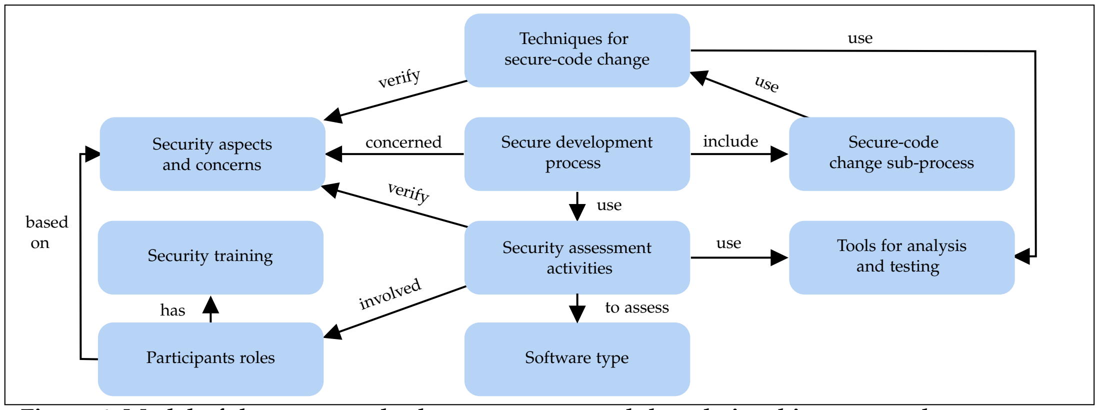
Diversity of security issues. The participants described different processes, activities, and tools for security assurance and different processes and techniques for ensuring secure code change. The activities, tools, and techniques were developed to address the different security issues (e.g., design-level security needs, language-based vulnerabilities, implementation of cryptographic solutions issues) that the developers and security experts encounter when developing secure software. Therefore, each organization chooses a set of assessment activities, techniques, tools, and processes to address the security issues that their security experts encounter when developing a secure software for their businesses. This is expressed well by participant **P9** who stated, when we asked them *"what should be done to ensure the security of code change?"* with *"I don't think I have ever seen anything the silver bullet"*.

Table 7: Position and role.

Positions	Roles	# Participants
Developer	Write the code Run unit tests Perform code analysis	10
Security team	Provide guideline for secure coding standards Develop security training Monitor the the code for vulnerabilities Assist the developers in addressing vulnerabilities Evaluate the risk of using new technologies	6
Architect	Integrate security in the design Support the developers in fixing security vulnerabilities	5
QA	Perform the required security tests	3
Risk Manager	Assess security risks Approve/Reject requests to deploy software	3
Technology asset owner	Own the given application	3
Project manager	Manage the given project	2
Audit team	Perform audit Archive code analysis results	1
Change manager	Perform impact analysis	1
Development manager	Provide technical or input feedback	1

Effectiveness of security assurance tools. Most of the participants (eight) stated that they use static code analysis tools. These tools, according to the participants, have, however, a high rate of false positives. This problem demotivates the developers on using the tools because they would need to spend a lot of time filtering the issues that the tools report. Participant **P9** said *"source code analysis has a very high false-positive rate have a lot of noise which developers hate."*

Ignoring human factors. Organizations tend to focus on the technical aspects and ignore the human factors although developing secure software requires the collaboration of people involved in the process. For instance, participant **P9** formulated that as follows: *"I think people, especially security people, IT people, in general, tend to choose technology, there are no people, there is no relation or feeling. Those are [...] the very few things that you have to consider when you put [..] an Appsec program"*. Effective communication is essential to the success of a secure change process. The developers would try to bypass the quality control gate (which is the security code analysis) if they find it difficult, ineffective, increase the workload, and


Techniques for secure-code change
use
verify
use
Security aspects and concerns
concerned
Secure development process
include
Secure-code change sub-process
verify
use
based on
Security training
Security assessment activities
use
Tools for analysis and testing
involved
has
to assess
Participants roles
Software type
Figure 6: Model of the secure-code change concepts and the relationships among these concepts.

may delay their projects–and may delay software releases. Security experts should get feedback from the developers about the effectiveness of the code analysis tools they employ, communicate with the other stakeholders and try to improve the processes, the guidelines, the tools, or the training of the developers.

5 DISCUSSIONS

This section summarizes the results of the study and discusses the impacts and limitations of the study.

5.1 Summary

Figure 6 shows the themes/concepts extracted from the study and the relationship between these themes/concepts. The figure shows that the secure development process includes a secure-code change sub-process. The processes are commonly used for several application types including Web applications and control system software. The people participating in the process assume specific roles and may have appropriate training that is developed based on the security aspects of concern to the organizations. Some of the participants in the secure development process and secure-code change sub-process use a set of tools to assess the security of the software. The assessment activities and techniques for both process and sub-process verify that the aspects and concerns are addressed.

We observe from the study that code review is the most used technique to ensure the secure-code change (8 participants) followed by code analysis and penetration testing, each is practiced by five participants. We expected that many organizations would use keywords search but we found only one participant (**P11**). In addition, we observe that three participants:**P2**, **P7**, and **P9** perform the three techniques: code review, code analysis, and penetration testing. These three participants mentioned that they have/had training on developing secure software.

5.2 Impact of the Study

There are extensive work on techniques, tools, and processes to develop a secure development process, including the development of secure software using the agile approach [5] and identification of characteristics of vulnerable code change [7]. This study enumerates the security aspects that the developers address when they modify their software.

The study revealed that dependency vulnerabilities, authentication and authorization, and OWASP top 10 are the main aspects that the participants consider when they change secure-software,[4] Improving the techniques and tools to address these aspects should have important impacts on efficient and effective maintenance of secure software.

Recall that the main techniques that practitioners use to ensure secure-code change are: code review, code analysis, penetration testing, and keywords search. We observe that most of the participants rely on code analysis and penetration testing to ensure the security of modified software. They limit the use of manual code review to security-sensitive code segments: the segments that are flagged by the keywords search algorithms. This use of security assessment tools is risky. For example, the assessment may fail to detect a change that breaks the authorization mechanisms simply because the new code does not include the keywords.

In addition, we found that the organizations that develop Web applications ensure the security of their modified software but the organizations that develop control systems do not. The community should raise awareness of the companies that develop cyber-physical systems about the impact of security vulnerabilities.

5.3 Threats of Validity

We discuss in the following the limitations of the study, which we classify into construct validity, internal validity, conclusion validity, and external validity threats [8, 29].

Construct validity. This category concerns the threats to the relation between the performed study and the goal of the study. To address this validity, we performed a literature review, designed an interview protocol and tested it with some experts. In addition, we collected the information from eleven participants who have different roles and are located in different cities, which gives confidence in the stability of the collected data.

Internal Validity. The validity concerns the relationship between the study and its results. At the beginning of each of

[4]They are invoked by most of the participants–see Table 3

the interviews, we tell the interviewee the aim of the interview, which should help in ensuring that the participants and interviewer share the same goal. The diversity of the roles of the participants may, however, impact the relationship between the study and results.

Conclusion validity. This validity concerns the ability to draw correct conclusions from the results of the study. To address these threats, the main author discussed the codes and themes with the second author to reduce the subjectivity of the results.

External Validity. External validity concerns the generalization of the outcomes of the study. The participants in the study were security experts and developers in eleven organizations. While most of them are from the software engineering industry, we believe that they bring diverse experience that supports generalizing the results.

6 CONCLUSION

This paper reports about the current practices on secure-code change in the industry. In this study, we found that organizations (four out of eleven) provide formal training of their developers on developing secure software. We conclude that (1) there are six security aspects and concerns; dependency vulnerabilities, authentication and authorization, access control, OWASP top 10, encryption, and buffer overflow, (2) the common security assessment activities are static code analysis, dynamic code analysis, object code analysis, software composition analysis, penetration testing, and code review, (3) there are four techniques to ensure secure-code changes; review, code analysis, testing, and keyword search, and (4) there are eight people involved in the process namely, developer, development manager, project manager, risk manager, security team, audit team, QA, and architect.

ACKNOWLEDGMENT

The authors thank Mike Johnson and Jim McClurg from John Deere for the thorough discussions along the execution of the research. They also thank the anonymous participants in the study for their contributions. This project is funded partially by a grant from John Deere.

REFERENCES

[1] [n. d.]. OWASP Top 10-2017. https://www.owasp.org/index.php/Top_10-2017_Top_10.

[2] [n. d.]. Source Sode to Object Code Traceability Study - adacore. https://www.adacore.com/uploads/books/pdf/traceability-sample.pdf.

[3] 2019. What Is a Buffer Overflow? Learn About Buffer Overrun Vulnerabilities, Exploits & Attacks. https://www.veracode.com/security/buffer-overflow.

[4] M. Abdelkhalek, L. Ben Othmane, and A. Jamil. 2019. Identification of the Impacts of Code Changes on the Security of Software. In Proc. of the IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), Vol. 2. 569–574.

[5] L. ben Othmane, P. Angin, H. Weffers, and B. Bhargava. 2014. Extending the Agile Development Process to Develop Acceptably Secure Software. IEEE Transactions on Dependable and Secure Computing 11, 6 (Nov 2014), 497–509.

[6] A. Bosu. 2014. Characteristics of the Vulnerable Code Changes Identified Through Peer Code Review. In Companion Proceedings of the 36th International Conference on Software Engineering (ICSE Companion 2014). 736–738.

[7] A. Bosu, J. C. Carver, M. Hafiz, P. Hilley, and D. Janni. 2014. Identifying the Characteristics of Vulnerable Code Changes: An Empirical Study. In Proc. of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014). 257–268.

[8] Daniela S. Cruzes and Lotfi ben Othmane. 2017. Empirical Research for Software Security: Foundations and Experience. Taylor & Francis Group, LLC, Chapter Threats to Validity in Software Security Empirical Research, 275–300.

[9] IEEE Cybersecurity. 2015. Understand How Integrating External Components Changes Your Attack Surface. https://cybersecurity.ieee.org/blog/2015/11/13/understand-how-integrating-external-components-changes-your-attack-surface/.

[10] W. Du. 2019. Computer Security: A Hands-on Approach. Wenliang Du. https://books.google.com/books?id=spOJxAEACAAJ

[11] P. H. Engebretson and D. Kennedy. [n. d.]. The basics of hacking and penetration testing. Syngress/Elsevier.

[12] A. Ghahrai. 2018. Static Analysis vs Dynamic Analysis in Software Testing. https://www.testingexcellence.com/static-analysis-vs-dynamic-analysis-software-testing/.

[13] M. Hilton, N. Nelson, T. Tunnell, D. Marinov, and D. Dig. 2017. Trade-offs in Continuous Integration: Assurance, Security, and Flexibility. In Proc. of the 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2017). 197–207.

[14] M. Kim and D. Notkin. 2009. Discovering and Representing Systematic Code Changes. In Proc. of the 31st International Conference on Software Engineering (ICSE '09). 309–319.

[15] Bixin Li, Xiaobing Sun, Hareton Leung, and Sai Zhang. 2013. A survey of code-based change impact analysis techniques. Software: testing, verification and reliability 23, 8 (December 2013), 613–646.

[16] A. Meneely, A. C. Rodriguez Tejeda, B. Spates, S. Trudeau, D. Neuberger, K. Whitlock, C. Ketant, and K. Davis. 2014. An Empirical Investigation of Socio-technical Code Review Metrics and Security Vulnerabilities. In Proc. of the 6th International Workshop on Social Software Engineering (SSE 2014). 37–44.

[17] S. F. P. Mohamed, Fauziah. Baharom , A. Deraman, J.Yahya, and H. Mohd. 2016. Secure software practices among Malaysian software practitioners: An exploratory study. AIP Conference Proceedings 1761, 1 (2016), 020086. https://doi.org/10.1063/1.4960926

[18] L. B. Othmane and A. Ali. 2016. Towards Effective Security Assurance for Incremental Software Development the Case of Zen Cart Application. In Proc. of the 11th International Conference on Availability, Reliability and Security (ARES). 564–571.

[19] OWASP. 2017. OWASP Top 10 Application Security Risks – 2017. https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf.

[20] OWASP. 2017. OWASP Top 10 Application Security Risks – 2017. https://www.owasp.org/index.php/Top_10-2017_A5-Broken_Access_Control.

[21] Eric S. Raymond. 2001. The cathedral & the bazaar: musings on Linux and open source by an accidental revolutionary. OReilly.

[22] R. Rymon. 2019. Software Composition Analysis Explained.

[23] J. Saldana. 2015. The coding manual for qualitative researchers. Sage Publications.

[24] R. S. Sandhu and P. Samarati. 1994. Access control: principle and practice. IEEE Communications Magazine 32, 9 (Sep. 1994), 40–48.

[25] A. Sharma, P. S. Grover, and R. Kumar. 2009. Dependency Analysis for Component-based Software Systems. SIGSOFT Softw. Eng. Notes 34, 4 (2009), 1–6.

[26] E. t Mougoue. 2019. What is the secure software development life cycle (SDLC)?: Synopsys. https://www.synopsys.com/blogs/software-security/secure-sdlc/

[27] D.W. Turner. 2010. Qualitative interview design: a practical guide for novice investigators. The Qualitative Report.

[28] R. Vanciu and M. Abi-Antoun. 2013. Finding architectural flaws using constraints. In Proc. of the 28th IEEE/ACM International Conference on Automated Software Engineering (ASE). 334–344.

[29] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Bjöorn Regnell, and Anders Wesslén. 2000. Experimentation in Software Engineering: An Introduction. Kluwer Academic Publishers, Norwell, MA, USA.

[30] A. T. T. Ying, G. C. Murphy, R. Ng, and M. C. Chu-Carroll. 2004. Predicting source code changes by mining change history. IEEE Transactions on Software Engineering 30, 9 (Sep. 2004), 574–586.