Masthead Logo

Electrical and Computer Engineering Conference Papers, Posters and Presentations

Electrical and Computer Engineering

2019

Identification of the Impacts of Code Changes on the Security of Software

Moataz Abdelkhalek
Iowa State University, moataz@iastate.edu

Ameerah-Muhsina Jamil
Iowa State University, amjamil@iastate.edu

Lotfi ben Othmane
Iowa State University, othmanel@iastate.edu

Follow this and additional works at: https://lib.dr.iastate.edu/ece_conf

Part of the Electrical and Computer Engineering Commons, Information Security Commons, and the Software Engineering Commons

Recommended Citation

Abdelkhalek, Moataz; Jamil, Ameerah-Muhsina; and ben Othmane, Lotfi, "Identification of the Impacts of Code Changes on the Security of Software" (2019). *Electrical and Computer Engineering Conference Papers, Posters and Presentations*. 78.
https://lib.dr.iastate.edu/ece_conf/78

This Conference Proceeding is brought to you for free and open access by the Electrical and Computer Engineering at Iowa State University Digital Repository. It has been accepted for inclusion in Electrical and Computer Engineering Conference Papers, Posters and Presentations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.

Identification of the Impacts of Code Changes on the Security of Software

Abstract
Companies develop their software in versions and iterations. Ensuring the security of each additional version using code review is costly and time consuming. This paper investigates automated tracing of the impacts of code changes on the security of a given software. To this end, we use call graphs to model the software code, and security assurance cases to model the security requirements of the software. Then we relate assurance case elements to code through the entry point methods of the software, creating a map of monitored security functions. This mapping allows to evaluate the security requirements that are affected by code changes. The approach is implemented in a set of tools and evaluated using three open-source ERP/Ecommerce software applications. The limited evaluation showed that the approach is effective in identifying the impacts of code changes on the security of the software. The approach promises to considerably reduce the security assessment time of the subsequent releases and iterations of software, keeping the initial security state throughout the software lifetime.

Disciplines
Electrical and Computer Engineering | Information Security | Software Engineering

Comments
This is a manuscript of the proceeding Abdelkhalek, Moataz, Ameerah-Muhsina Jamil, and Lotfi ben Othmane. "Identification of the Impacts of Code Changes on the Security of Software." *COMPSAC 2019: Data Driven Intelligence for a Smarter World* (2019): 1-6.

Rights
© 2019 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

This conference proceeding is available at Iowa State University Digital Repository: https://lib.dr.iastate.edu/ece_conf/78

Identification of the Impacts of Code Changes on the Security of Software

Moataz Abdelkhalek, Ameerah-Muhsina Jamil, and Lotfi ben Othmane

Iowa State University, Ames, IA, USA

Abstract— **Companies develop their software in versions and iterations. Ensuring the security of each additional version using code review is costly and time consuming. This paper investigates automated tracing of the impacts of code changes on the security of a given software. To this end, we use call graphs to model the software code, and security assurance cases to model the security requirements of the software. Then we relate assurance case elements to code through the entry point methods of the software, creating a map of monitored security functions. This mapping allows to evaluate the security requirements that are affected by code changes. The approach is implemented in a set of tools and evaluated using three open-source ERP/E-commerce software applications. The limited evaluation showed that the approach is effective in identifying the impacts of code changes on the security of the software. The approach promises to considerably reduce the security assessment time of the subsequent releases and iterations of software, keeping the initial security state throughout the software lifetime.**

I. Introduction

The economic impacts of cyber-security have led companies to adopt secure software development practices [1], including security assurance: ensuring the software fulfills its security requirements. The common approach to identify code-level vulnerabilities, such as buffer overflow, is to use code analysis techniques [2], which can parse large software in reasonable time. Logic flaws, such as broken authentication and broken access control, which account for 50% of the vulnerabilities [3], [4] cannot be detected by code analysis techniques.

```
1  public void changedClass
2  {
3    public void writeToFile(String myInput)
4    {// Do Something
5     FileIOPermission myPerm =
6     new FileIOPermission(PermissionState.Read);
7        //   Change to: Unrestricted
8     myPerm.Demand();
9     // Do Something
10    }}
```

Listing 1. An example of risky code change.

Assume that we have a `login` function that uses the `writeToFile` method from class `changedClass` of Listing 1 and assume that a developer changes the value of `PermissionState` used in the `FileIOPermisison` from `Read` to `Unrestricted`. The common security assessment tools do not detect the impacts of this code change because the change impacts the logic, not the structure. The security experts may not detect the problem using manual review of the security-critical code chunks, the `login` function code, because the change is not in the `login` function itself, but in one of its dependencies. Failure to detect the impact of this code change can lead to serious privilege escalation vulnerabilities.

Code changes are changes to the software code that may introduce new functionality to the software, fix existing defects, or adapt the software to changes in its environment, cf. [5]. These changes can break the security of the software. Currently, there are two common approaches to identify code changes that impact the security of a software. The first approach requires the senior developers to discuss the security impacts of each change request before making the change. This approach is frequently used, especially for open source software although it is time consuming [4]. The second approach is to have annotations in the code itself to mark the security aspects related to the given code portion. The annotations warn the developers who change the code of these components about the implication of these changes on the security of their software [6].

The used approaches are time-consuming and prone to omission and commission errors. First, they consider security aspects in a conceptual way and not as mechanisms. For instance, they consider the impact of code changes on access control and not how access control may fail due to code change. Second, the approaches rely on the understanding of the experts of the interactions between the software components; that is, on the architecture view the developers make based on their experience with the code of the software.

This paper investigates automated tracing of the impacts of code changes on the security of the given software. To this end, we (1) model the software code using the call graph technique, which represents the calling relationships between the functions/methods of the software, (2) model the security assurance of the software using security assurance cases, and (3) relate assurance case elements to the code through code entry points, which compose the attack surface of software.

The contributions of the paper are:

1) Propose an approach to trace the impact of code

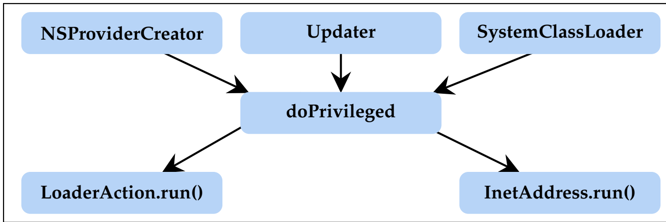

Fig. 1. Simplified part of generated call graph of OpenZ ERP application.

changes on the security of the given software.

2) Develop a set of tools that implement the proposed approach for secure code changes.
3) Evaluate the proposed approach and tool-set using three open-source software.

The paper is organized as follows. First, we give a primer on security assurance and call graph in Section II and discuss related work in Section III. Next, we describe the proposed approach in Section IV and the components of the system in Section V. Then, we report about the results of evaluating the approach on three open-source software in Section VI and conclude the paper in Section VII.

II. Background

This section gives an overview on security assurance cases and call graphs.

A. Assurance Cases

Security assurance is the degree of confidence that the security requirements are satisfied [7]; it is the level of trust that the software meets its security specifications and does not perform unintended functions that compromise its security [8].

The security assurance of a software could be described using a *security assurance case*: a semi-formal graphical representation of a collection of security-related claims, arguments and pieces of evidence that can prove that a software exhibits a security property [9], [10]. Security *claims*, aka security goals, are high-level security requirements. The *pieces of evidence* are the results of the verification of the claims through, for example, security tests, security code reviews, and proofs. An *argument* is a justification that evidence sets support the related claim.

The Goals-Structuring Notation (GSN) is a common way to represent assurance cases [11]. It is a graphical argumentation notations that represent the claims, evidence sets, arguments, and the relationships between them in a tree structure. Figure 3 illustrates an example of a partial security assurance case, which will be explained in Section VI.

B. Call Graphs

A program *call graph* is a compile-time data structure that represents the run-time calling relationships among a program's methods or functions [12], [13]. Each node of the graph represents a method/function and each edge that links two nodes represents a call relationship between these two methods/functions. Call graphs are commonly used in inter-procedural static analysis including compilers, verification tools and program analysis tools [14]. Programs often include pointers and conditions for methods/functions calls. The call graph tools consider such program complexities to ensure accurate construction of call graphs from programs [12].

Figure 1 shows an example of a simplified small section of a call graph obtained from analyzing one of our test applications. We observe in this example that method `doPrivileged()` is called by methods `SystemClassLoader()`, `Updater()`, and `NSProviderCreator()` and calls methods `LoaderAction.run()` and `InetAddress.run()`.

III. Related Work

The common approaches for software security assurance focus on ensuring that the security properties are enforced at each version of the software. They have a significant cost associated with their use in time and resources [15], [4]. This led to the idea to investigate the evolution of security requirements and design of a software, with the assumption that the code of the software represents the updated requirements and design [16]. Jurjens et al. [15], for example, extended the UML models with security annotations, called UMLSec, and investigated the use of design models to support the multiple evolution paths and the incremental verification process of the evolving models.

Bosu and Carver conducted an empirical evaluation of peer-code review of Open Source Software (OSS) to detect security vulnerabilities [17]. They developed a Java application to mine the text related to the security flaws from the code review comments and stored the mining results in a database. They query the database using specific keywords and inspect results manually. They found that code review indeed helps in identifying and removing security vulnerabilities. In another work, Bosu et. al. analyzed the security vulnerabilities that could be discovered by code review, identified the characteristics of vulnerable code, and the developers that wrote it [18]. They found that the common types of identified vulnerabilities are buffer overflow and cross-site scripting. In addition, they found that the number of line changes also affects the likelihood of vulnerabilities and modified files have higher likelihood to have vulnerabilities than the new ones.

Rapid Release Change (RCC) had attracted researchers in a way that frequent releases of new features are likely to increase the number of vulnerabilities and the discovery rate. Clark et al. [19] investigated the relation between RCC adapted by Firefox and the different security vulnerabilities. Firefox, which adapted RCC, releases a new version every 6 weeks. The authors found that vulnerabilities are disclosed in newer code

as often as the older code. The authors also found that frequent releases increase the time needed by attackers to learn the software code, which forces them to adapt their tools to cope with the changes.

Vanciu and Abi-Antoun mentioned in [6] that half of the security vulnerabilities are caused by the flaws in the architecture of the system. They proposed Scoria, a semi-automated approach to find architectural flaws that can be used during Architectural Risk Analysis (ARA). The main features of Scoria are the use of static analysis to extract from the code, the use of queries to assign security properties to objects and edges, and the use of dataflow to trace code from edges that the constraints highlight that potential architectural flaws exist. The results of the study prove Scoria ability to detect security vulnerabilities that cause architectural flaws. However, the focus was only on detecting information disclosure and tampering vulnerabilities.

Othmane et al. [20] proposed integrating security re-assurance into the agile software development processes to ensure the security of the developed software with each iteration. In addition, they demonstrated the use of the technique to iteratively develop security features that fulfill their security requirements [21]. The process helps, for example, to identify customer change requests that conflict with the security requirements of the iteration. In a subsequent work, they analyzed the impacts of incremental secure software development for an existing open source software, Zen Cart [4].

The existing solutions for security assurance in the literature require full reassessment or are adhoc, e.g., using keywords search, which can be ineffective. This paper proposes tracing the code changes to the security requirements that they affect, if any. This should reduce the cost of security reassurance and maintain the security of the software throughout its lifetime.

IV. Approach

The approach hypothesizes that code changes could be traced to the security requirements. To identify the impacts of code changes on the security of software, we model (1) the software code, (2) the security requirements of the software and (3) the relationship between the software code and its security requirements. The entry points are the source of data that flow into software [22]. Thus, security requirements are strongly related to the entry points of the software, since attacks happen by sending or receiving data to the system [23] using channels, such as open ports, connection sockets, and input forms. Hence, if code changes could be tracked to entry points, having a code path from the given entry point to the changed code portion, would indicate if the change is security-related. A security-related change can then be flagged for intensive reviewing.

Several techniques are used to implement the proposed approach. First, we represent software code with call graphs, which represent the calling relationships between the functions/methods of the software. We used the open-source static code analysis tool T. J. Watson Libraries for Analysis (WALA) [24] to generate the call graph of a given software. For software analysis, WALA was chosen because it (1) produces low overhead in performance and time, (2) supports several programming languages including Java and JavaScript and (3) can be extended to support more languages.

Algorithm 1 Identify code changes that impact the security of the software

```
    Input CallGraph
    Input AssuranceCase
    Input FunctionAssuranceCaseMap
    Output SuspiciusCodeChanges
  procedure DetectSuspiciousChanges
     ChangeList ← IdentifyChangedFunctions()
     TopNode
     ← getMonitoredNodes(FunctionAssuranceCaseMap)
     while ChangeList ≠ ∅ do
        changedNode ← ChangeList.pop()
        PathsList ← getPathList(changedNode,TopNode)
        if (PathsList ≠ ∅) then
            SuspiciusCodeChanges
            ← SuspiciusCodeChanges ⋃ changedNode
        Return SuspiciusCodeChanges
     End
```

Second, we model the security requirements of the software using security assurance cases. The assurance cases are used at the design phase of the software to specify graphically the security requirements of the software and choose the security mechanisms that address them. In addition, in the testing phase, it can be used in the security assessments of the software to verify the security of the software.

Third, the assurance case elements are manually related to the corresponding methods of the software. This mapping requires expertise knowledge of the software and its security requirements. In this phase, developers and security administrators should be involved to ensure correct and accurate mapping.

Fourth, as developers use version control systems such as `Git` [25] to trace code changes; to ensure alignment in our toolset; we used integration with the `Git` versioning system to track and detect code changes. Then, these changes were mapped to corresponding functions and classes.

These four main steps allow us to detect and map the code changes to the assurance cases that reflect the security requirements of the given software. As depicted by Algorithm 1, the `DetectSuspiciousChanges` mechanism monitors the storage of the software for new versions using `Git`, detects the code changes, and identifies the methods/functions that the changes

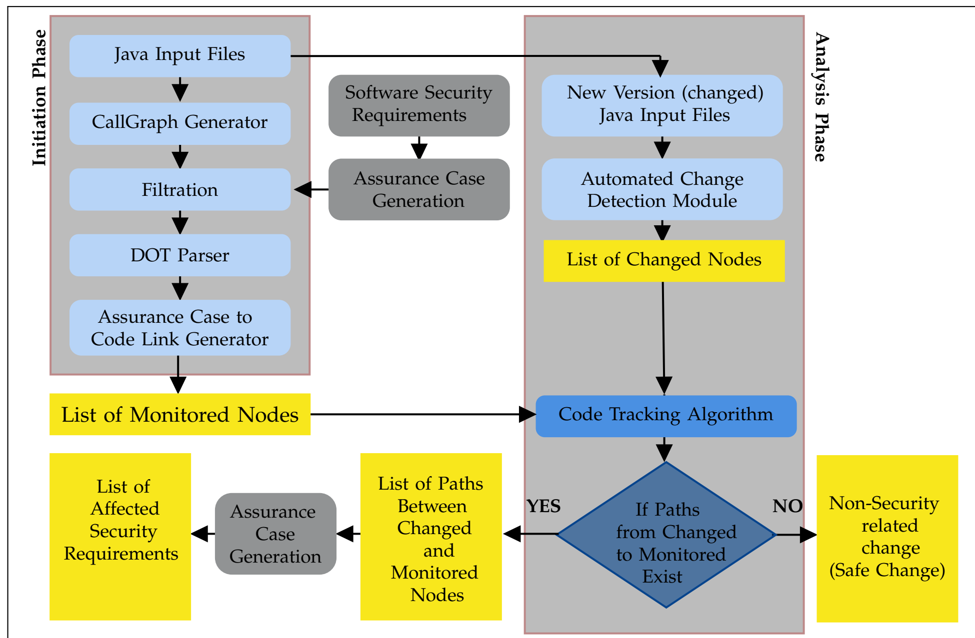

Fig. 2. Components of the System

belong to. Then, it uses the call graph to identify the paths that link the identified methods/functions to the ones associated with the security assurance claims of the software. Changes related to assurance cases are flagged as suspicious changes that require comprehensive review to ensure they do not break the security requirements of the software.

V. System overview - Secure code change analysis toolset

We developed a set of tools to assist developers and security experts to create call graphs, design security assurance cases, and map the security claims and evidence parts of the assurance case to the related entry point methods of the given software. Figure 2 shows the interactions between the components of the approach, as discussed in the previous section.

We developed a graphical tool to design security assurance cases for the software [26] as an Eclipse plug-in. The tool supports adding and removing security claims, arguments or rational, evidence sets, and strategy nodes to a security assurance case, storing the assurance case in an XML format for further use. In addition, we developed a module that creates the call graph of the given software using WALA [24] and produces a call graph file in the DOT format that can be later parsed using graphical tools.[1] With the help of the graphical call graphs, the previously developed assurance case elements could be associated to corresponding methods/functions of the software marking them as monitored nodes in the call graph. To automatically detect changes, we also developed a module that continuously queries the software `Git` repository to identify the delta cumulative changes of any two versions of a given software.

Finally, we developed a *Code Tracking Algorithm* that searches all the possible paths between the detected changed nodes and the monitored nodes; the ones associated with the security assurance cases. Then, the changed methods included in the identified paths are flagged as suspicious and the claims associated with the related monitored methods are flagged as potentially broken and should be reviewed manually.

VI. Evaluation of the proposed approach

We selected three software applications to evaluate our approach, which are: Compiere,[2] Shopizer,[3] and OpenZ.[4]. The used selection criteria are: (1) written in Java, (2) be an open-source security-sensitive software, (3) completeness of the software source code, (4) having recent and regular updates, and (5) have high frequency of downloads. Table I provides the details of the selected software applications.

We applied the workflow of Figure 2 for the three software applications. In the first phase, the initiation phase, we created the security assurance cases of the software based on the Payment Card Industry (PCI) [28] and the OWASP top 10 requirements [29]. These choices were made because the chosen software applications are financial and the original software security requirements were not available. Figure 3 shows the security assurance case for the safe authentication and session management of Compiere application. Next, we created

[1]The call graph file could be visualized using tools, such as Graphviz [27].

[2]https://sourceforge.net/projects/compiere/
[3]https://github.com/shopizer-ecommerce/shopizer
[4]https://sourceforge.net/projects/openz/

TABLE I

LIST OF SELECTED OPEN-SOURCE SOFTWARE.

Application	Lines of Code	Function Calls	Classes	Avg Methods/Class	Last Updated	Last version
OpenZ	1727793	846364	3456	19	Sep 2018	3.5.0
Compiere limited version	22059	4257	136	7	Nov 2016	1.3.0
Compiere Full version + Oracle XE	649571	154917	2289	15	Mar 2017	Community Edition 3.5
Shopizer	106858	26705	961	6	Dec 2018	2.3.0

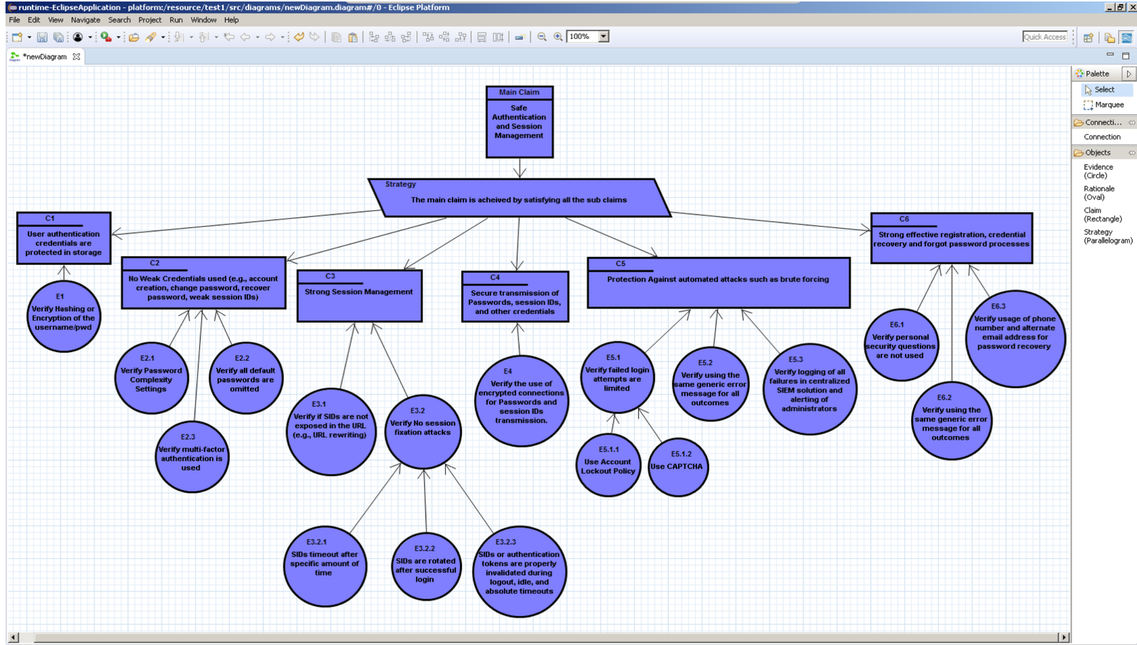

Fig. 3. Assurance Case for session management in Compiere application

the call graphs for each of the software using software analysis module. Table II provides the performance of the code analysis and call graph generation module. After that, we associated the methods to their related security claims. Table III gives a simplified map that we developed for the session management claims for Compiere application. This phase resulted into a set of security-sensitive methods that the system needs to monitor.

The second phase of the workflow, as depicted by Figure 2 consists of capturing the impacts of code changes on the security of the software. To increase the number of test-cases, we applied the analysis on a large number of changes using different versions of the software applications, in addition to, intentionally injected code changes. Then, we applied the identification of code changes tool and the code tracking algorithm to identify whether the code changes are associated with any of the security-sensitive entry-point methods.

The results showed that the toolset could detect all the security related changes successfully. In addition, the overhead of using the toolset was very low, as it processes software that has a call graph with up-to 500 nodes in 1 seconds, up-to 1000 nodes in 20-50 seconds, and up-to 2500 nodes in 5 minutes. The call graph generated with the tool was filtered using keywords to select the security-sensitive portions of the code, to improve the performance of the analysis.

TABLE II

PERFORMANCE MEASUREMENT OF THE CALL GRAPH GENERATION TOOL.

	Initiation Phase	
Application	No. of nodes	Time
OpenZ	846364	9 min 36 sec
Compiere limited version	4257	11 sec
Compiere Full version + Oracle XE	154917	2 min 14sec
Shopizer	26705	20 sec

Although, the evaluation of the approach is limited to three software applications, we experimented with a good number of test-cases, which showed that the approach is effective in identifying the impacts of code changes on the security of the software. The approach promises to considerably reduce the security assessment time of the subsequent releases and iterations of software, keeping the initial security state throughout the software lifetime. We intend to improve the usability of the toolset and extensively evaluate it using more commonly used security-sensitive software.

TABLE III

Map of the security sub-claims of the session fixation claim and their entry point methods in Compiere application.

Assurance Case Element	Associated Methods
User authentication credentials are protected in storage	util.Login.checkPermission()
No weak credentials used	security.AccessControl Context.checkPermission()
Strong session management	process.SessionStartAll.clinit() process.SessionEndAll.main()
Secure transmission of Passwords, session IDs, and other credentials	security.PermissionsHash.add() db.TestConnection()
Protection against automated attacks such as brute forcing	apps.ProcessCtl() process.SvrProcess() process.FactAcctReset()
Strong effective registration, credential recovery and forgot password processes	DB.Oracle() model.Registration()

VII. Conclusion

We developed a toolset that trace the impacts of code changes on the security of a given software and evaluated it using three open-source software applications. We found that the proposed approach successfully identifies the security impacts of code changes and proves our hypothesis. It needs, though, a thorough evaluation with commonly used software to assess its capabilities and limitations. The approach would help to considerably reduce the security code review time and the time that developers spend discussing the impacts of the proposed code changes on the security of the software.

Acknowledgment

The authors thank Mike Johnson and Jim McClurg from John Deere for the thorough discussions along the execution of the research. This project is funded by a grant from John Deere.

References

[1] J. Viega and G. McGraw, "Building secure software: How to avoid security problems the right way," 2006.

[2] M. Pistoia, S. Chandra, S. J. Fink, and E. Yahav, "A survey of static analysis methods for identifying security vulnerabilities in software systems," *IBM Syst. J.*, vol. 46, pp. 265–288, Apr. 2007.

[3] I. Arce, J. DelGrosso, D. Dhillon, C. Kern, T. Kohno, C. Landwehr, G. McGraw, B. Schoenfield, M. Seltzer, D. Spinellis, I. Tarandach, and J. Wes, "Avoiding the top 10 of software security design flaws." = https://www.computer.org/cms/CYBSI/docs/Top-10-Flaws.pdf, 11 2015.

[4] L. ben Othmane and A. Ali, "Towards effective security assurance for incremental software development the case of zen cart application," in *2016 11th International Conference on Availability, Reliability and Security (ARES)*, pp. 564–571, Aug 2016.

[5] M. Acharya and B. Robinson, "Practical change impact analysis based on static program slicing for industrial software systems," in *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pp. 746–755, 2011.

[6] R. Vanciu and M. Abi-Antoun, "Finding architectural flaws using constraints," in *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 334–344, Nov 2013.

[7] M. D. Abrams and P. R. Toth, "A head start on assurance," in *Proc. of an Invitational Workshop on Information Technology (IT) Assurance and Trustworthiness*, (Williamsburg, Virginia), March 1994.

[8] S. Katzke, "Security assurance: Does anybody care?," in *Proc. of the 21st National Information Systems Security Conference*, (Arlington, VA, USA), Oct 1998.

[9] J. Goodenough, H. F. Lipson, and C. B. Weinstock, "Arguing security - creating security assurance cases." = https://www.us-cert.gov/bsi/articles/knowledge/assurance-cases/arguing-security-creating-security-assurance-cases, Nov. 2014.

[10] H. F. Lipson and C. B. Weinstock, "Evidence of assurance: Laying the foundation for a credible security case." = https://www.us-cert.gov/bsi/articles/knowledge/assurance-cases/evidence-assurance-laying-foundation-credible-security-case, Nov. 2014.

[11] T. Kelly and R. Weaver, "The goal structuring notation – a safety argument notation," *Proc. Dependable Systems and Networks - Workshop on Assurance Cases*, July 2004.

[12] D. Grove and C. Chambers, "A framework for call graph construction algorithms," *ACM Trans. Program. Lang. Syst.*, vol. 23, pp. 685–746, Nov. 2001.

[13] B. G. Ryder, "Constructing the call graph of a program," *IEEE Transactions on Software Engineering*, vol. 5, pp. 216–226, May 1979.

[14] O. Lhoták, "Comparing call graphs," in *Proceedings of the 7th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, PASTE '07, pp. 37–42, 2007.

[15] J. Jürjens, L. Marchal, M. Ochoa, and H. Schmidt, "Incremental security verification for evolving umlsec models," in *Proc. of the 7th European Conference on Modelling Foundations and Applications*, ECMFA'11, (Birmingham, UK), pp. 52–68, 2011.

[16] "Securechange security engineering for lifelong evolvable systems." `http://www.securechange.eu/`. accessed on Jan. 2019.

[17] A. Bosu and J. C. Carver, "Peer code review to prevent security vulnerabilities: An empirical evaluation," *2013 IEEE Seventh International Conference on Software Security and Reliability Companion*, 2013.

[18] A. Bosu, J. C. Carver, M. Hafiz, P. Hilley, and D. Janni, "Identifying the characteristics of vulnerable code changes: an empirical study," *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2014*, 2014.

[19] S. Clark, M. Collis, M. Blaze, and J. M. Smith, "Moving targets: Security and rapid-release in firefox," *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security - CCS 14*, 2014.

[20] L. ben Othmane, P. Angin, H. Weffers, and B. Bhargava, "Extending the agile development process to develop acceptably secure software," *IEEE Transactions on Dependable and Secure Computing*, vol. 11, pp. 497–509, Nov 2014.

[21] L. Ben Othmane, P. Angin, and B. Bhargava, "Using assurance cases to develop iteratively security features using scrum," in *Proc. of Ninth International Conference onAvailability, Reliability and Security (ARES), 2014*, (Fribourg, Switzerland), pp. 490–497, 2014.

[22] C. Theisen, H. Sohn, D. Tripp, and L. Williams, "Bp: Profiling vulnerabilities on the attack surface," in *2018 IEEE Cybersecurity Development (SecDev)*, pp. 110–119, Sep. 2018.

[23] P. K. Manadhata and J. M. Wing, "An attack surface metric," *IEEE Transactions on Software Engineering*, vol. 37, pp. 371–386, May 2011.

[24] "T.J. Watson Libraries for Analysis." `http://wala.sourceforge.net/wiki/index.php/Main_Page`. accessed in Jan 2019.

[25] "Git." `https://git-scm.com/`. accessed in Jan. 2019.

[26] "Tool for designing security assurance cases." `https://github.com/lbenothmane/SecAssuranceCase`. accessed in Jan 2019.

[27] "Graphviz - graph visualization software." `https://www.graphviz.org/`. accessed in Jan 2019.

[28] "Payment card industry (pci) data security standard (dss)." `https://www.pcisecuritystandards.org/document_library`. accessed in Jan 2019.

[29] "The open web application security project OWASP." `https://www.owasp.org/`, Accessed on May. 2016.