J.Natn.Sci.Foundation Sri Lanka 2020 48 (1): 93 - 98
DOI: http://dx.doi.org/10.4038/jnsfsr.v48i1.8950

RESEARCH COMMUNICATION

A security specific knowledge modelling approach for secure software engineering

A Abeyrathna[1], C Samarage[1], B Dahanayake[1], C Wijesiriwardana[2*] and P Wimalaratne[1]
[1] *University of Colombo School of Computing, No 35, Reid Avenue, Colombo 07.*
[2] *Faculty of Information Technology, University of Moratuwa, Katubedda, Moratuwa.*

Submitted: 26 March 2019; Revised: 03 June 2019; Accepted: 27 September 2019

Abstract: The paradigm shift of '*Build Security In*' has emerged in recent decades with the underpinning idea that software security has to be an integral part of all the phases of the software development lifecycle. As a result, each phase of the lifecycle is associated with security specific best practices such as threat modelling and static code analysis. It was observed that various artefacts (i.e., security requirements, architectural flaws, bug reports, security test cases) generated as a result of security best practices tend to be disconnected from each other. This creates a significant barrier to ensure that the security issues identified in the architectural level are incorporated in the implementation level. In order to address this issue, this paper presents a knowledge-modelling based approach to semantically infer the associations between architectural level security flaws and code level security bugs, which is manually tedious. Threat modelling and static analysis are used to identify security flaws, and security bugs, respectively. The case study based experimental results reveal that the architectural security flaws have a significant impact on originating security bugs in the code level.

Keywords: Security touchpoints, software security, static code analysis, threat modelling.

INTRODUCTION

Security vulnerabilities in software can imperil intellectual property, consumer trust, and business operations and services (Wang *et al*., 2009). The consequences of security breaches lead to both tangible (e.g., loss of business and productivity) and intangible (e.g. customers trust) losses. Having identified the critical need for software security, the paradigm shift of 'Building Security In' has emerged in the recent decades (Erlingsson, 2016; Karim *et al*., 2016; Piessens & Verbauwhede, 2016). This paradigm shift requires software security to be addressed in all phases of the software development lifecycle. Literature reveals that most security vulnerabilities result from defects that are unintentionally introduced in the software during the design phase and the development phase (McGraw, 2006). McGraw (2006) has identified code reviews and architectural risk analysis as the top two best practices to minimise the security vulnerabilities in software systems. These best practices called security touchpoints are associated with the artefacts produced during the implementation phase (i.e., code base) and the design phase (i.e., design documents), respectively in the traditional software development lifecycle. Even in organisations with mature software development processes, software artefacts created end up to be disconnected from each other (Antoniol *et al*., 2000). Furthermore, to the best of our knowledge, existing tools are not capable of identifying security-specific associations between the artefacts generated in each phase of the lifecycle. This incapability reveals a critical research gap of semantically interlinking the artefacts originated at the implementation phase of the design phase. This paper presents a conceptual framework and a proof-of-concept implementation to semantically interlink architectural level security flaws and code level security bugs. Security flaws are identified based on STRIDE (Hernan *et al*., 2006) threat categorisation

* Corresponding author (chaman@uom.lk; https://orcid.org/0000-0002-1124-425X)

model introduced by Microsoft, which helps to identify threats from the attackers' perspective by classifying attackers' goals into 06 threat categories. Security bugs are determined based on OWASP Top 10 (Wichers, 2013) vulnerabilities, which is the ten most critical web application security risks providing a great awareness for web application security. Security flaws and bugs are interlinked by employing a knowledge-modelling based technique, which facilitates inferring the associations that are manually tedious.

METHODOLOGY

This approach aims at inferring the associations between security flaws and security bugs that are introduced during the design phase and the implementation phase of the software development lifecycle. As stated previously security flaws are identified in terms of STRIDE threat categorisation and security bugs are represented regarding OWASP Top 10 vulnerabilities. The approach consists of three main constituents: identifying security flaws, identifying security bugs, and inferring relationships among flaws and bugs.

Identifying and pre-processing security flaws

Security flaws are identified through an architectural risk analysis, which includes explicitly identifying security risks in the software architecture that were produced in the design phase of the software development. In this paper, threat modelling has been used as the architectural risk analysis method due to several noteworthy reasons such as the ability to work with high-level design diagrams, simplicity to employ in different contexts, and explicit tool support. For example, the threat-modelling process can be initiated by drawing a data flow diagram (DFD). According to Abi-Antoun *et al*. (2007), architectural level security flaws can be effectively identified by analysing Level 0 or Level 1 DFDs. The threat modelling process consists of three steps: decomposition, determination and ranking of threats, and countermeasures and mitigation.

Decomposition: This step is concerned with gaining an understanding of the application and how it interacts with external entities. This knowledge helps in identifying entry points to see where a potential attacker could interact with the application.

Determination and ranking of threats: In this step, threats are determined and categorised according to a threat categorisation methodology. The goal of threat categorisation is to identify threats from both the attacker's perspective and defensive perspective.

Countermeasures and mitigation: In this step, mitigations, and countermeasures are identified for the ranked threats.

Using static analysis to identify security bugs

Static analysis is a well-known software engineering best practice that is used to detect the security bugs that appear in the source code of a software system. For adequately understanding the code level security bugs, they are categorised based on OWASP Top 10 vulnerabilities. OWASP Top 10 is the ten most critical web application security risks, which provide a powerful awareness document for web application security. Although the static analysis detects the bugs that are categorised into OWASP vulnerabilities, that information is not sufficient to generate a relationship with security flaws. On the other hand, the benefits of the existing static analysis tools in isolation for security specific analysis of multiple large-scale software systems is highly questionable (Wijesiriwardana & Wimalaratne, 2017). Therefore, it was decided to utilise OWASP Proactive Controls (*www.owasp.org*), which is a set of developer-centric security techniques that can be included in every software project. The OWASP Top 10 vulnerabilities have been mapped to proactive controls as suggested in UcedaVelez and Morana (2015).

Inferring relationships among flaws and bugs

Inferring logical relationships between security flaws and bugs are expected to obtain *via* STRIDE and OWASP. However, STRIDE mainly focuses on the attacking perspective of software security. On the other hand, Application Security Frame (ASF) is a threat categorisation model, which helps to identify the threats from the defensive perspective. For an in-depth analysis of the threats affecting the software application data and functional assets, both the attacker view and the defensive view for the enumeration of threats were considered as essential. As stated previously, the threat-modelling process only focuses on attacker's perspective. Therefore, to involve the defensive standpoint, a relationship has been identified between STRIDE and ASF. As indicated earlier, a knowledge base is used to identify the association between threat categories and bug categories through a semantic text similarity matching model. The set of countermeasures of ASF and summarised proactive control descriptions were used to get the semantic similarity between ASF and proactive controls.

Semantic text similarity between ASF and proactive controls

The semantic text similarity was calculated for every single security control in ASF with every single proactive control. The descriptions of ASF security controls and proactive controls are not limited to a single phrase. Accordingly, the semantic text similarity of each phrase of the description of a particular ASF security control was calculated concerning each phrase of the description of proactive control. Consequently, by borrowing the semantic similarity concepts presented in Han *et al*. (2013), the average semantic similarity score between a specific ASF security control (Ai) and proactive control (Pi) was calculated as follows.

$$SemS(Ai, Pi) = \left[\left(\sum_{i=0}^{nm} Vi\right) \div nm\right]$$

where:

Ai : description of ASF having *n* phrases

Pi : description of proactive controls having *m* phrases

Vi : similarity between phrase *i* of ASF and phrase *i* of proactive control

Knowledge base

It contains the facts and rules related to the STRIDE, ASF, OWASP T10, proactive controls and semantic similarity scores between ASF and proactive controls. A frame-based approach was used for knowledge representation of facts (Merritt, 2012). Below is the structure of the frame for STRIDE.

Listing 1: Frame for STRIDE categories

```
frame (stride,
        [category_model – [val threat],
         types - val [spoofing, tampering, repudiation,
        information disclosure, denial of service,
        elevation of privileges]]]).
```

Eleven prolog rules were developed to infer the association between STRIDE and OWASP T10. Due to the space limitations only two rules are listed here together with a short description of each.

Rule 1 - Querying the knowledge base

```
isCausedByThreatCategories(BugCategory, TList_Unique) :-
                    findall(T, isCausedByThreatCategory(
            BugCategory, T), TList),
                        sort(TList, TList_Unique)
```

Rule 1 is used to query the knowledge base. The list of unique threat categories can be discovered by querying the knowledge base using a bug category.

Rule 2 - Discovering associated threat category using the bug category

```
isCausedByThreatCategory(BugCategory , T) :-
        lacksProactive(BugCategory , P), mapsToSecurity-
        Control(P , S),
        isWeakendByThreatCategory(S , T)
```

Rule 2 discovers the associated threat category utilising the bug category. The threat category was revealed using the subsequent rules on the right-hand side. The lacks Proactive (BugCateogry, ProactiveControl) was used to discover the proactive controls violated due to the given bug category. The association results given by the knowledge base were used to create the associations between bugs and threats.

Proof-of-concept implementation

In this research, as a proof-of-concept, a security analysis framework called Conexus has been implemented to infer the relationships between design and implementation artefacts by adhering to the principles described previously. Conexus framework has four main constituents: threat-based processing, bug-based processing, association inferencing, and knowledge base. The source code of the implementation can be found in a public Github repository *https://github.com/ashanthi93/SecurityFramework*.

Threat-based processing

Conexus required the user to draw the Level-0 or Level-1 DFD of a particular software application to generate the Threat Model. MS TMT (*www.microsoft.com/en-us/sdl*) is used for this purpose, and it creates a threat model, which includes threats categorised according to STRIDE categorisation. The threat model generated from MS TMT was obtained as an XML, which is further processed to extract threats. After that, the extracted threats were converted into threat objects, which contains the relevant details of risks introduced by this component.

Security bug-based processing

As of now, Conexus framework analyses the source code using SonarQube (*www.sonarque.org*), which is a widely used tool for continuous code quality improvements. The vulnerabilities identified as security bugs were categorised into OWASP T10. The bugs were further processed and classified into bug category objects. Bug category objects adhere to OWASP T10. Then the bug category objects were passed to association inference module.

Association inference module

Association loader was used for querying the knowledge base using the bug category objects to identify the associated threat categories. A prolog converter was built in SWI-Prolog to communicate with Java. Each bug category was used to query the knowledge base, and the associated threat type results were held inside the association loader. The associated threat type results and the bug objects were sent to the association linker. Threat category objects from STRIDE transformer and associated threat types and bug objects from association loader were the input to the association linker. Accordingly, the association objects were generated.

Knowledge base

The knowledge base was built using the SWI-Prolog. All the facts and rules aforementioned are contained in the knowledge base. The knowledge base has the capability of revising when the OWASP T10 or proactive controls are revised. On the other hand, knowledge base explicitly allows expanding the knowledge contained in it using the additional knowledge of security experts.

RESULTS AND DISCUSSION

The main focus of the evaluation was to find whether the potential root causes of an identified security bug lie in the design phase of the software system. A case study based evaluation was employed for the evaluation process of this approach.

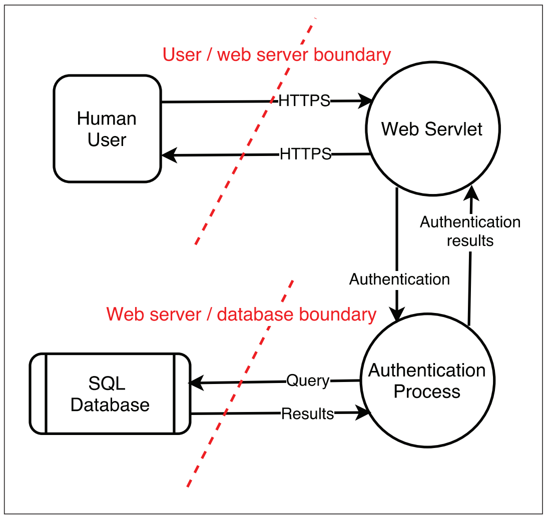

Figure 1: DFD of the user authentication component of the web application

Table 2: Summary of the identified threats in the web application

Threat type	Number of threats
Spoofing	6
Tampering	4
Repudiation	4
Information disclosure	2
Denial of service	9
Elevation of privilege	8

Case study: User authentication component of a web-based application

Figure 1 presents the DFD of the user authentication in a web-based application. It consists of two processes, one external entity, and a single data store together with associated data flows. In the evaluation process, threat modelling is conducted to identify the architectural-level security flaws, and static analysis is used to capture the security bugs at the implementation level. The association derived between security bugs and the threats are based on the security bug categories and threat categories. Table 2 depicts the possible threats identified by the threat modelling process. Similarly, through static code analysis, A2, A5, and A6 categories of OWASP have been captured. A1 to A10 are known as the top ten latest threat categories identified by OWASP. Then, the highly relevant causes (threat categories) of the security bug categories were identified, and the corresponding countermeasures were applied to remove the security bugs in the source code.

After repeating static analysis, it was observed that previously designated A2, A5 and A6 bug categories were removed successfully. Hence, it was evident that removing the potential security specific root causes at the design level leads to the security bugs at the code level. However, the accuracy of the results obtained from the experiments depends on the analysis outputs given from SonarQube, MS TMT as well as the semantic similarity scores.

The associations derived from the Conexus framework depicted the possible causes for a security bug. It was observed that, even with such an association, still it is difficult to pin-point the exact location of the source code because the DFD was a Level 0 diagram. Therefore, it is crucial to identify the lower level DFDs for efficient capturing of security bugs. The framework can be used conjointly with the re-engineering process of a previously developed software where the association between security flaws and security bugs can be identified and necessary actions taken to implement security in the software. Implementation of a software application sometimes includes the use of legacy software components. Thus, the Conexus framework can be used to ensure the security of the legacy components, which requires significant security improvements. In a security-focused agile development environment, Conexus framework can be used to ensure the security of the working software in each product increment.

CONCLUSIONS

This paper presents a knowledge modelling-based approach for finding traceability links between security bugs in the source code with the security flaws in the design phase. The association is derived by identifying threat categories and bug categories based on well-known classification schemes. The results of this study evidently show the fact that architectural level security issues lead to security bugs in the code base. On the other hand, the conceptual approach of this research could be applied to interlink various other software artefacts to discover unthought-of-yet-interesting hidden relationships of software systems.

The framework could serve as a stepping stone to the researchers in the field of software security, which was lacking previously. On the other hand, the knowledge base has provisions to evolve with different security aspects. As future work, the experiments are expected to repeat for large-scale open source software systems. Furthermore, it is planned to improve this research to directly interlink security bugs with security flaws by utilising attack trees or case-based reasoning.

Acknowledgments

The authors gratefully acknowledge the financial support provided by the National Research Council of Sri Lanka (Grant No: NRC 15-74).

REFERENCES

Abi-Antoun M., Wang D. & Torr P. (2007). Checking threat modeling data flow diagrams for implementation conformance and security. *Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering*, Atlanta, USA, pp. 393–396.
DOI: https://doi.org/10.1145/1321631.1321692

Antoniol G., Canfora G., Casazza G. & De Lucia A. (2000). Information retrieval models for recovering traceability links between code and documentation. *Proceedings of the IEEE International Conference on Software Maintenance*, San Jose, USA, 11–14 October, pp. 40–49.
DOI: https://doi.org/10.1109/ICSM.2000.883003

Erlingsson U. (2016). Data-driven software security: models and methods. *29th Computer Security Foundations Symposium (CSF)*, Lisbon, Portugal, 27 June–1 July, pp. 9–15.
DOI: https://doi.org/10.1109/CSF.2016.40

Han L., Kashyap A.L., Finin T., Mayfield J. & Weese J. (2013). UMBC_EBIQUITY-CORE: semantic textual similarity systems. *Second Joint Conference on Lexical and Computational Semantics (SEM),* Atlanta, USA, volume 1, pp. 44–52.

Hernan S., Lambert S., Ostwald T. & Shostack A. (2006). Threat modeling-uncover security design flaws using the stride approach. *MSDN Magazine-Louisville*, pp. 68–75.

Karim N.S.A., Albuolayan A., Saba T. & Rehman A. (2016). The practice of secure software development in SDLC: an investigation through existing model and a case study. *Security and Communication Networks* **9**(18): 5333–5345.
DOI: https://doi.org/10.1002/sec.1700

McGraw G. (2004). Software security. *IEEE Security and Privacy* **2**(2): 80–83.
DOI: https://doi.org/10.1109/MSECP.2004.1281254

Merritt D. (2012). *Building expert systems in Prolog*. Springer Science and Business Media. Berlin, Germany.

Piessens F. & Verbauwhede I. (2016). Software security: vulnerabilities and countermeasures for two attacker models. *Proceedings of the 2016 Conference on Design, Automation and Test in Europe*, 14–18 March, Dresden,

Germany, pp. 990–999.
DOI: https://doi.org/10.3850/9783981537079_0999

UcedaVelez T. & Morana M.M. (2015). *Risk Centric Threat Modeling: Process for Attack Simulation and Threat Analysis*. John Wiley & Sons, Hoboken, USA.
DOI: https://doi.org/10.1002/9781118988374

Wang J.A., Wang H., Guo M. & Xia M. (2009). Security metrics for software systems. *Proceedings of the 47th Annual Southeast Regional Conference*, pp. 1–6.
DOI: https://doi.org/10.1145/1566445.1566509

Wichers D. (2013). Owasptop-10 2013. OWASP Foundation.

Wijesiriwardana C. & Wimalaratne P. (2017). On the detection and analysis of software security vulnerabilities. *IEEE International Conference on IoT and Application (ICIOT)*, 19–20 May, Nagapattinam, India, pp. 1–4.
DOI: https://doi.org/10.1109/ICIOTA.2017.8073635