An Assurance Case Approach for Software Code Security

Ryota Miyabayashi
Nagoya University
Nagoya, Japan
miyabayashi.ryouta@g.mbox.nagoya-u.ac.jp

Noritoshi Atsumi
Kyoto University
Kyoto, Japan
atsumi.noritoshi.5u@kyoto-u.ac.jp

Shuji Morisaki
Nagoya University
Nagoya, Japan
morisaki@is.nagoya-u.ac.jp

Shuichiro Yamamoto
Nagoya University
Nagoya, Japan
yamamotosui@icts.nagoya-u.ac.jp

***Abstract*— The vulnerability of the security of the software codes has often caused by invalid input. Therefore, software security greatly depends on the safety of module inputs. In this paper, we propose a method to ensure the code security by building Assurance cases based on the relationship between input specifications and code portions of each module. In addition, a case study is accomplished to apply the proposed method for the published vulnerability of an OSS. The result shows that the method can detect the vulnerability of the OSS by the absence of necessary evidence in the developed assurance case based on the input specifications and portions of the OSS code.**

Keywords—Assurance case, code review, input vulnerability, security

I. Introduction

Reusing software code has the opportunity to develop software effectively and improve the quality of software developed. The reusing code shall be ensured to satisfy security and safety conditions, otherwise reusing code will cause threats for the target software. In case of existing codes, it is necessary to develop assurance cases for ensuring code security of the target code of reuse. If there were models for the target code of reuse, assurance cases could be developed based on models corresponding to the code by using model based assurance case development methods. If there was no models for the target code of reuse, it is necessary to develop assurance case directly in the code. Unfortunately, the assurance case development method based on code has not been established so far. In this paper, an approach to develop assurance case for existing code is proposed. Then an experiment is executed to develop the assurance case for the Open SSL code based on the corresponding specification. The result shows the effectiveness of the proposed method. The rest of the paper is as follows. Section II describes the background of assurance case development to ensure security of codes. Section III proposes an approach to develop assurance cases for code security. An experimental evaluation of the approach is described in section IV. Section V discusses the effectiveness of the approach. Section VI shows related work. Section VII concludes this paper.

II. Background

The objective of the approach is to clarify the method to develop assurance cases for codes that have no related model. There was a problem that the corresponding models often had been disappeared from the case of existing codes. Therefore, the assurance case development method based on models are not be able to apply for the existing codes. For example, open source software (OSS) specifications and codes have been published, but it is unusual for OSS models exist. We assume that there is a specification for the code as the target of assurance. In case of developing assurance case, we focus on the input and output of the specification. For example, SSL/TLS Protocol V 1.0 is the specification for the OpenSSL 1.0 code. Unfortunately, the OpenSSL 1.0 code had the vulnerability on the FREAK attack. FREAK attack stands for "Factoring RSA Export Keys (FREAK)." FREAK is the attack to exploit the weak RSA encryption key used in the 1990s. The cause of this vulnerability came from the omission of code portions for inputs of functions to manipulate the RSA encryption key.

The code shall have portions for the input and output specified in the specification. Otherwise, the code has the possibility of omissions from the point of input and output. This notion can be called as Defect Detection by Evidence (DDBE). The evidence is the appropriate code portions related to the specific part of the specification.

Assurance case is widely used to ensure that system has the expected property, such as safety, security and dependability. Assurance case is defined by using claim, strategy, context, and evidence. The Claim is decomposed by strategy into sub claims. Context explains assumptions of claims and reasons of the claim decomposition. Evidence is used to show why claims hold. GSN, Goal Structuring Notation, is a method to describe assurance cases [3, 4]. Assurance case development methods based on models have been proposed [5, 6, 13, 14, 15]. Assurance case for security also have been proposed [7, 8]. However, these assurance development methods assumed models to exist. Code based assurance case development approach has not been proposed so far. This paper proposes an approach to develop assurance cases without assuring models to exist for the code.

III. Apprach to develop assurance case for code

The evidence based assurance case development approach is described below.

[Input] specification and code

[Output] assurance case with evidence

[Method] Develop assurance case with evidence based on specification and code by the following steps.

(STEP1) Define target claims on input and output parameters by the specification. Create a top goal as the claim, "the code is valid for the specification." And create the context node which name is "code and specification." Then create the connection from the top claim to the context node.

(STEP2) Develop assurance case by structuring sub claims based on the top claim as follows.

(STEP2-1) Decompose the top claim into two sub claims by using strategy node which has the name "Explain by parameters." The names of sub claims are "Parameters are valid" and "parameter relationships are valid." And create the context node which has the name "Parameters and their relationships." Then create the connection from the strategy to the context node. The context node shows the reason of the decomposition.

(STEP2-2) Decompose the node "Parameters are valid" into two sub claims by using the strategy node "Explain by parameter types." The names of the sub claims are "Input parameters are valid," and "Output parameters are valid." Two context nodes "Input Parameter constraint" and "Output Parameter constraint" are created and connected to the claims above, respectively.

(STEP2-3) Decompose the node "Parameter relationships are valid" into three sub claims by using the strategy node "Explain by parameter relationship types." The names of the sub claims are "Input parameter relationships are valid," "Input output parameter relationships are valid," and "Output parameter relationships are valid." Two context nodes "Input Parameter relationship constraint," "Input output Parameter relationship constraint," and "Output Parameter relationship constraint" are created and connected to the claims above, respectively.

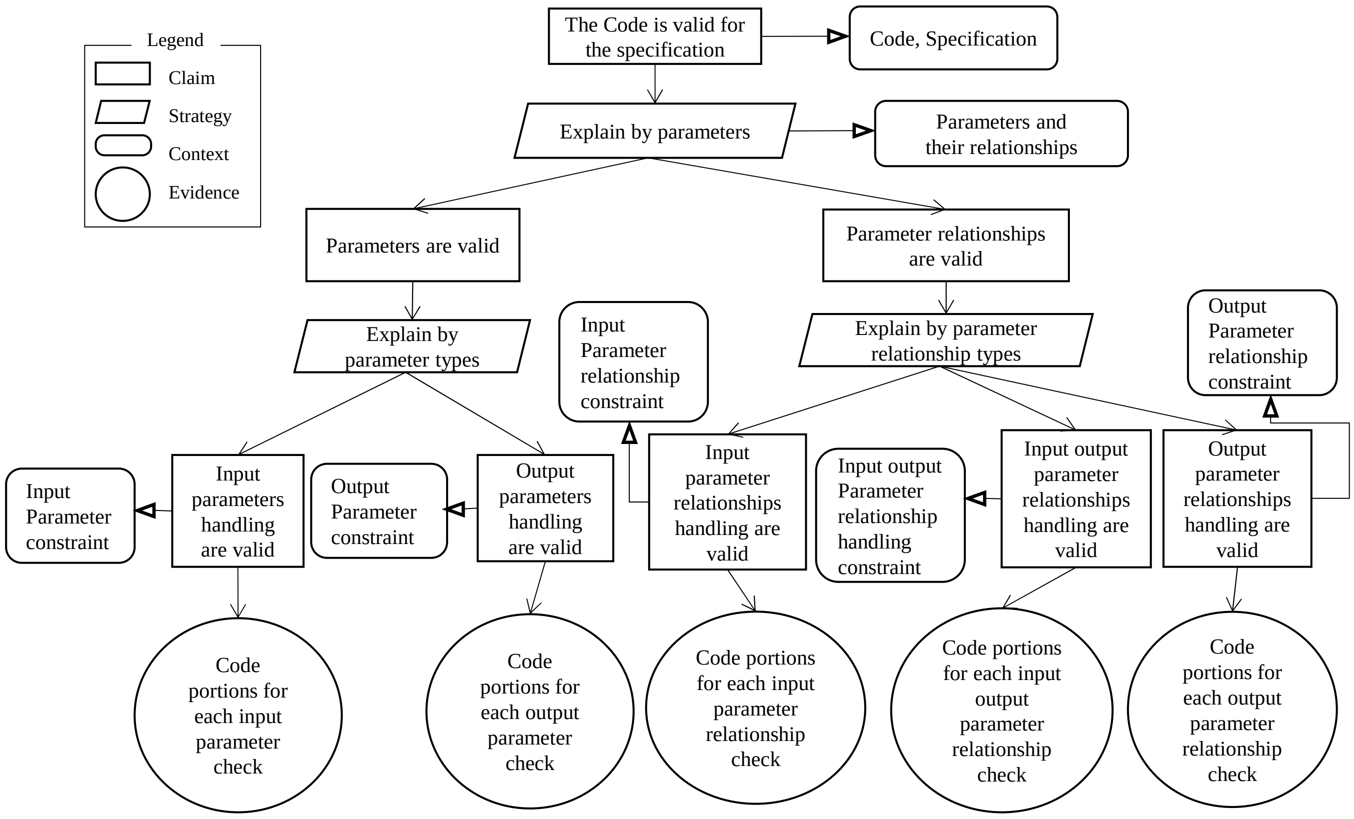
(STEP3) The bottom level claims developed in STEP 2-2 and 2-3 are called TBE (To Be Explained) claims. Explore code portions related to TBE claims.

(STEP4) Create evidence nodes related to the detected code potions. Then create decompositions from the TBE claims to the evidence nodes, respectively.

(STEP5) If code portions as evidence are not discovered in the target code, the corresponding target TBE claims have no evidence. In this case, these TBE claims show the defect of the code. This means the code is invalid for the specification according to TBE claims that have no evidence.

(End of method)

The method described above shows an approach to find defects of the code according to the specification. Fig. 1 shows the created assurance case template based on the method.



Legend
Claim
Strategy
Context
Evidence
The Code is valid for the specification
Code, Specification
Explain by parameters
Parameters and their relationships
Parameters are valid
Parameter relationships are valid
Explain by parameter types
Input Parameter relationship constraint
Explain by parameter relationship types
Output Parameter relationship constraint
Input Parameter constraint
Input parameters handling are valid
Output Parameter constraint
Output parameters handling are valid
Input parameter relationships handling are valid
Input output Parameter relationship handling constraint
Input output parameter relationships handling are valid
Output parameter relationships handling are valid
Code portions for each input parameter check
Code portions for each output parameter check
Code portions for each input parameter relationship check
Code portions for each input output parameter relationship check
Code portions for each output parameter relationship check

Fig. 1. Assurance case template created by the method

IV. EXPERIMENT

An experiment was planned to evaluate the effectiveness of the approach to develop assurance cases based on codes proposed above. The overview of the experiment target code and specification are as follows. The specification is the SSL/TLS Protocol V 1.0 written in 3584 lines in English. The code is OpenSSL 1.0.1j s3_clnt.c [1] written in 3469 lines in C language. The subject of the experiment is one undergraduate student of Nagoya University. The subject first extracted 5 input parameters shown in Table.1. The time to analyze input parameters from the specification was 12 hours.

TABLE I. INPUT PARAMETERS

Input parameters	Constraints
Session Id	is equivalent to the Id from client
Compression Id	is a compression Id sent from client
Encryption suit	is an encryption Id sent from client
Certificate information	has no error
Key	is appropriate to the encryption suit

Then the subject described the assurance case without evidence based on the SSL/TLS Protocol V 1.0. There were 11 TBE claims. The time to develop the assurance case was 5 hours.

After the extraction of TBE claims, the subject then tried to discover code portions from OpenSSL 1.0.1j s3_clnt.c as evidence for 11 TBE. There were portions of code for 10 TBE claims. There was no portion of code for one TBE claim.

The portion of the assurance case for the unexplained TBE is shown in Fig.2.

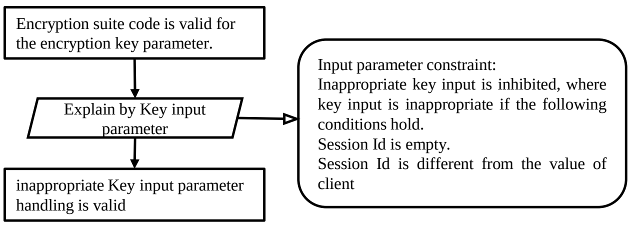

Encryption suite code is valid for the encryption key parameter.
Explain by Key input parameter
inappropriate Key input parameter handling is valid
Input parameter constraint:
Inappropriate key input is inhibited, where key input is inappropriate if the following conditions hold.
Session Id is empty.
Session Id is different from the value of client

Fig. 2. Portion of assurance case for the missin TBE

The portion tried to explain the claim "Encryption suite code is valid for the encryption key parameter." The claim is decomposed to the sub claim "inappropriate Key input parameter handling is valid" by the strategy "Explain by Key input parameter." The strategy clarifies the input parameter constraint as shown in the context connected to the strategy. The input parameter constraint are as follows.

Inappropriate key input is inhibited, where key input is inappropriate if the following conditions hold.

Session Id is empty.

Session Id is different from client sent value

In this case, the expected code portion is to check the Key input parameter conditions as described in the input parameter constraint and handle the exception if the Key input parameter fails to satisfy the condition. Therefore, the evidence for the TBE is the code portion corresponds to inhibit inappropriate Key input parameter. However, the code portion for the TBE was not able to discover in the OpenSSL code. Moreover, the missing TBE was corresponded to the vulnerability of OpenSSL known as FREAK (Factoring RSA Export Keys) [2]. The time to find code portions for 11 TBE was 10 hours.

A. *Overview of FREAK*

The overview of FREAK is as follows [2].

(1) The client side sends the encryption list to the server side. The encryption list does not include the export grade RSA encryption.
(2) The attacker falsifies the encryption list that includes the export grade RSA encryption only.
(3) The server side selects an encryption method from the sent encryption list. The selected encryption is export grade encryption, because the falsification of the encryption list by the attacker.
(4) In case that the export grade encryption is selected, the server sends a 512 bit RSA key as a temporal key to the client. As the key is often reused, an attacker is able to know the key beforehand. Moreover, the secret key is also able to get easily, because the key size is 512 bits.
(5) The attacker rewrites the encryption suit to the encryption in the original encryption list. The server sends the rewritten encryption suit to the client. If the encryption suit was not rewritten, the error is detected by the check of encryption suit.
(6) As the client receives the key, the client encrypts the premaster secret by using the key and sends it to the server. The premaster secret is the basic data of common key for the subsequent secure communication. The premaster secret information should not be known to another one.
(7) The attacker decrypts the premaster secret by using a secret key that corresponds to known RSA temporal key. The attacker can know the common key to encrypt the subsequent secure communication.
(8) The attacker falsifies and reads the contents of the secure communication by using the common key.

The cause of FREAK is as follows.

Although the client does not request the export grade RSA encryption, the client uses the received key for encryption. In case of general request for encryption, the export grade RSA encryption is not included in the encryption list. Therefore, the server side does not send a key to the client, but sends a certification to the client.

The defect found by the proposed approach is the non-existence of the key check code portion. The revision of the defect countermeasures the vulnerability of FREAK. In fact, the remedy of the OpenSSL is the addition of check code if the encryption suit corresponds to the export grade in case of the key is 512 bits RSA encryption. This showed the proposed approach can detect defects correspond to security vulnerability of codes.

V. DISUCUSSION

A. *Effectiveness*

As described in section IV, the proposed method has been applied to the client key exchange code. The TBE shows that key check is necessary. However, there was no code portion for the TBE. Therefore, we can conclude that there was an omission of key check in the target code. This shows that the proposed method is effective to discover defects of the code by using specifications.

B. *Work effort*

The working hour for the experiment was 27 hours in total as shown in Table1. The ratios of three kinds of works are 44.4%, 18.5%, and 37.1%, respectively.

TABLE II. HOURS OF TASKS

Task	Constraints	Hours
1	Input analysis from specification	12
2	Assurance case development and TBE claims identification	5
3	Discovery of code portions for TBE claims	10

The most time consuming work is the input analysis on the specification. The complexity of specification description style causes the problem. If the specification described in a concise fashion to understand input parameters easily, the work hours on input analysis will be reduced.

The work hours of the code portion discovery took 37.1% of the total effort. The difficulty of reading code by human subject causes the problem. If the code is clearly structured for understanding TBE claims, the work hours on the discovery will also be reduced.

The work hours to develop the assurance case and TBE claims was only 18.5% of the total effort. This shows that the proposed approach provides a common generic template for describing assurance case for ensuring security. The proposed approach provides a straightforward way to describe assurance cases for explaining TBE claims based on input parameters.

C. *Efficiency*

There are three activities for the proposed method.

A1) develop input parameters based on the specification

A2) develop assurance case based on input parameters and extract TBE claims

A3) discover code portions as evidence for each TBE claim

The work amount of A2 is not large, because assurance case can easily be developed by input parameters. The work amount of A1 was heavy for the case of OpenSSL specification, because the input information was embedded in the specification sentences and the explanations were very complex. If the specification structure was plain to extract input information, the work lord of A1 would extensively be improved. The work amount of A3 depends on the effort to understand the code. Code understanding is always a tough activity. In other words, if the code has a clear traceability relationship with TBE claims, the discovery of code portions of TBE claims will become an easy task by using the traceability relationship. This consideration implies an idea to develop codes based on TBE claims of assurance cases related to specifications. This idea will also make coding effort efficient.

D. *Applicability*

The assumption of the proposed method is that program code has inputs and the input controls the behavior of the code. However, practical programs have inputs. The method can be applied to security vulnerability analysis at the level of code. Omissions and errors on input cause many security problems. This shows the applicability of the proposed method. Moreover, the proposed method is not specific to security but is also able to apply to assure code implementations based on inputs and outputs.

E. *Sufficiency of the template*

If there is a code omission for manipulating parameters, the corresponding evidence of the template shall not been developed. This lack of the evidence for filling the template shows that the security claim for the code is not be supported for the code.

Suppose that the assurance case template developed from the code is complete with no lack of evidence. As there is no omission of code portions for checking input and output parameters, the code has no vulnerability caused by the lack of checking parameters.

F. *Limitation*

As the number of experiments was only one case, more experiments are necessary to show the effectiveness of the proposed method. The extraction of TBE claims from specifications depends on analysts. Therefore, reviews of extracted TBE claims are necessary to validate. Moreover, if the specification is incorrect, the extracted TBE claims are also incorrect. The review of the specification is also necessary before application of the proposed method. Finally, the discovery of code portions for TBE claims also depends on analysts. Analysts may fail to find the correct portions of code for TBE claims. Therefore, review of the matching between TBE claims and code portions is also necessary.

VI. RELATED WORK

There were researches to detect software vulnerability. Svace [9] is a tool to analyze codes statically based on dataflow. A case study [10] showed that approximately 20 % effort was able to reduce by static analysis tools. Although static analysis tools are effective, the false positive ratio of the static analysis tools is very high [11]. The more the static analysis tool detects vulnerability portions, the more work effort is necessary to ensure the correct portions related to the vulnerability. This shows the incompleteness and the inefficiency of the static analysis tools.

Chucky [12] is a method to help the analyst analyze codes statically by hands. The strong point of Chucky is the ability to decide defects by human static analysis. While their method is able to expose missing security checks effectively, it cannot assure that these missing checks lead to vulnerabilities.

The proposed method uses both specifications and codes to guide vulnerability analysis. The proposed code analysis uses input information and TBE claims of assurance case. The limited number of TBE claims makes the human analysis of codes efficient. Moreover, it greatly improves the false positive ratio, because TBE claims are correctly able to show the necessary code portions to prevent security vulnerability.

VII. CONCLUSION

In this paper, a defect detection method on codes by using assurance case was proposed. The method extracts TBE claims based on the specification for the code. Code portions are then discovered as evidence of the TBE claims. An experiment to evaluate the effectiveness of the approach has been described by applying the proposed approach for the Open SSL code. 11 TBE claims were extracted in the experiment. Although 10 TBE claims had been related to code portions, there was no code portion for one TBE claim that are corresponding to the code defect of security vulnerability known as FREAK. The result shows the proposed approach can discover code defects on security threats.

Future work includes more case studies for clarifying the effectiveness of the approach. Moreover, tool support is necessary to find code portions for TBE claims by analyzing data flow on parameters.

ACKNOWLEDGMENT

This work has been conducted as a part of "Research Initiative on Advanced Software Engineering in 2015" supported by Software Reliability Enhancement Center (SEC), Information Technology Promotion Agency Japan (IPA).

REFERENCES

[1] Openssl, https://www.openssl.org/, 2015.

[2] Freak: Factoring rsa export keys, https://www.smacktls.com/#, 2015.

[3] T. Kelly, “Arguing Safety, a Systematic Approach to Managing Safety Cases,” PhD Thesis, Department of Computer Science, University of York, 1998.

[4] T. Kelly, and R. Weaver, “The Goal Structuring Notation – A Safety Argument Notation,” Proceedings of the Dependable Systems and Networks 2004 Workshop on Assurance Cases, 2004.

[5] S. Yamamoto, and Y. Matsuno, “An evaluation of argument patterns to reduce pitfalls of applying Assurance Case,” Assure2013.

[6] S. Yamamoto, “An approach to assure Dependability through ArchiMate,” SAFECOMP 2015 Workshops, LNCS 9338, PP.50-61, Assure 2015, DOI: 10.1007/978-3-319-24249-1_5

[7] Shuichiro Yamamoto and Nobuhide Kobayashi, Mobile Security Assurance through ArchiMate, Vol. 4, No. 3 of IT Convergence Practice, pp.1-8, (INPRA), 2017, http://inpra.yolasite.com/vol4no3.php

[8] Yamamoto, S., Kaneko, T., and Tanaka, H., A Proposal on Security Case Based on Common Criteria, ICT-EurAsia 2013, pp.331-336

[9] V.P. Ivannikov, A.A. Belevantsev, A.E. Borodin, V.N. Ignatiev, D.M. Zhurikhin, and A.I. Avetisyan, Static analyzer svace for finding defects in a source program code, Programming and Computing Software., vol.40, no.5, pp.265‑275, Sept. 2014.

[10] M. Nadeem, B.J. Williams, and E.B. Allen, High false positive detection of security vulnerabilities: A case study, Proceedings of the 50th Annual Southeast Regional Conference, pp.359‑360

[11] D. Baca, B. Carlsson, and L. Lundberg, 2008. Evaluating the cost reduction of static code analysis for software security, Proceedings of the Third ACM SIGPLAN Workshop on Programming Languages and Analysis for Security, pp.79‑88

[12] F. Yamaguchi, C. Wressnegger, H. Gascon, and K. Rieck, Chucky: Exposing missing checks in source code for vulnerability discovery,” Proceedings of the 2013 ACM SIGSAC Conference on Computer &#38; Communications Security, pp.499‑510, CCS ’ 13, 2013

[13] G. Despotou, T. Kelly, Design and Development of Dependability Case Architecture during System Development, proceedings of the 25th International System Safety Conference (ISSC), System Safety Society, 2007

[14] R. Bloomfield and P. Bishop, Safety and assurance cases: Past, present and possible future – an Adelard perspective, procedings of the 18th Safety-Critical Systems Symposium, pp. 51-67, 2010.

[15] E. Denney and G. Pai, I. Habli, Perspectives on Software Safety Case Development for Unmanned Aircraft, proc. 42nd Annual IEEE/IFIP Intl. Conf. Dependable Systems and Networks(DSN2012), pp.1-8, 2012.