Assurance Carrying Code for Software Supply Chain

Yutaka Matsuno * , Yoriyuki Yamagata †, Hideaki Nishihara†, and Yuichiro Hosokawa ‡
* College of Science and Technology, Nihon University, Japan
Email: matsuno.yutaka@nihon-u.ac.jp
† Cyber Physical Security Research Center, National Institute of Advanced Industrial Science and Technology (AIST), Japan
Email: yoriyuki.yamagata@aist.go.jp, h.nishihara@aist.go.jp
‡ Department of Liberal Arts, Gunma Prefectural Women's University, Japan
Email: hosokawayuichiro@mail.gpwu.ac.jp

***Abstract*—Modern software systems are composed of software components supplied by a software supply chain, and it has become difficult to maintain the dependability of the software supply chain. To address this problem, we introduce *assurance carrying code*, a framework in which every software component in a software supply chain has its own assurance case. When integrating a software component into a supply chain, the stakeholders check (manually or automatically) the assurance case to determine whether or not the software component is dependable for the supply chain. We introduce a pattern language for Goal Structuring Notation (GSN) formalized by λ-calculus, which is used in a theory of functional programming languages theory.**

***Index Terms*—assurance cases, proof carrying code, formal languages, λ -calculus**

I. Introduction

Software systems have become increasingly complex. They contain various components including off-the-shelf software or open source modules, and connect to external systems such as cloud systems. Moreover, a system may be modified continuously. In such situations, it is very difficult to maintain control of the entire system.

One source of the difficulties is the software supply chain. In general, it is very costly to obtain information about acquired software. Third-parties are unwilling to share details of their products and their development processes (which may have already been lost), and thus the supply chain network becomes unclear and uncontrollable. Security risks occurring in supply chains have been discussed ([1], [2]), and incidents related to supply chains have been made public recently. In one incident, vulnerabilities were discovered in a software library applied to wide-range embedded systems over 20 years [3]. The library was allowed to modify or extend to adapt to users' products. Therefore, it was difficult to trace its history and see how wide-spread it is used in detail. In another incident, malicious software was distributed as regular updates [4]. Attackers tampered with update programs, and the programs were released on the manufacturer's download site. These incidents affected numerous systems including safety-critical systems, infrastructure controls, government information systems, and more. Subsequently, in May 2021, US President Biden signed an executive order that addresses supply chain security. Furthermore, NIST is revising its Cyber Supply Chain Risk Management Practices [5].

Assurance cases [6] support accountability in a supply chain, as assurance cases show clear arguments and evidence of the dependability of the supply chain. For software supply chains, we propose *Assurance Carrying Code*. In the framework, each software component in a supply chain has its own assurance case, and their assurance cases can be integrated into one case for the supply chain. To do so, we formalize assurance cases written in Goal Structuring Notation (GSN) [6] as a functional programming language, on the basis of λ-*calculus*.

The literature [6] uses the fundamental concepts of formal logic and English syntax in defining the GSN method to improve the expression of arguments. Meanwhile, prominent applications of λ-calculus include logic and linguistics, and more specifically, proof theory and Montague grammar. Particularly in the former, the correspondence between λ-calculus and natural deduction has been proven [7]. Thus, it would be reasonable to formalize assurance cases written in GSN in a functional programming language based on λ-calculus. Such an approach would open the way for connecting the safety arguments with key areas (logic, linguistics, and computer programming).

II. Assurance Carrying Code

Proof carrying code [8] is a software mechanism that allows a host system to verify properties of an application through a formal proof that accompanies the application's executable code. Using proof carrying code as our inspiration, we have been developing an assurance carrying code system, in which every software component has its own assurance case. When system developers integrate the code into a software supply chain, they can check the assurance case to determine whether the software code is safe (or secure) to install into the supply chain or not. The framework for assurance carrying code is shown in Fig. 1. As shown in the figure, each software component has its own GSN diagram (attached to the code or remotely on a database). When integrating a software

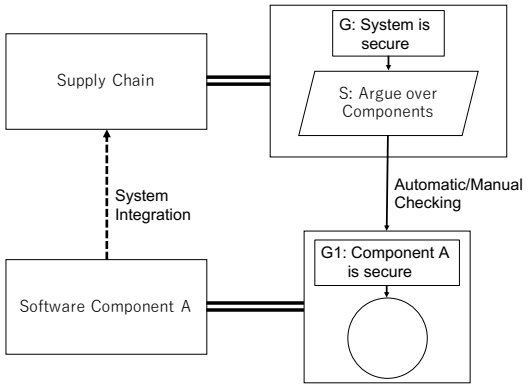

Fig. 1. A Framework of Assurance Carrying Code

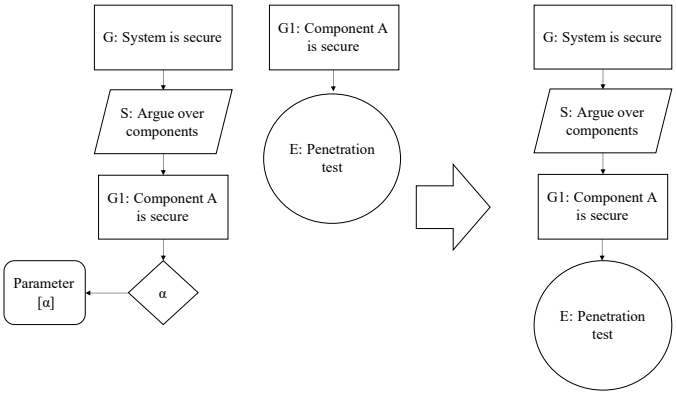

Fig. 2. Pattern reduction

component into a supply chain, the GSN diagram ("G1: Component A is secure") is automatically or manually checked by the assurance carrying code system, and if it is acceptable, the software component is integrated into the supply chain and the GSN diagram is also merged into the GSN diagram of the entire supply chain ("G: System is secure") .

We have been developing a web-based GSN tool [9] which can be used to remotely share and edit GSN diagrams using a GSN database. We are currently in the process of designing the framework (Fig. 1) using the tool.

III. A Pattern Language for GSN

To realize the assurance carrying code system, we propose a new pattern language which has graphical and term notations. Unlike Denny and Pai [10], our pattern language is a general purpose programming language based on λ-calculus, one of the basis of functional programming. Our language is based on λ-calculus and satisfies desirable properties such as *confluence*, which implies that any sequence of computations yields the same results if it terminates. Thus, the computation in our pattern language can be fully automated, in contrast to previous approaches [11]–[13] which require human guidance to obtain the final GSN diagrams.

Fig. 2 shows our notation for a GSN pattern and its reduction. The left-most diagram shows a pattern in which an argument for the goal G1 is abstracted away. When we provide an argument for the goal G1, for example, on the basis of a penetration test, the GSN pattern is instantiated into the GSN diagram in the right, which provides a full argument for the goal G ("System is secure"). In our framework, GSN diagrams and patterns are defined as follows.

$$P ::= \alpha \mid e[g] \mid S[g, g_1, \ldots, g_n] \mid \lambda\alpha.P \mid P_1 P_2 \qquad (1)$$

Here, α is a parameter for an argument, $e[g]$ is evidence for a goal g, $S[g, g_1, \ldots, g_n]$ is a strategy for deriving a goal g from goals $g_1, \ldots, g_n$. $\lambda\alpha.P$ behaves as a function which substitutes an argument Q to a parameter α when it is applied to another argument Q. $P_1 P_2$ is a pattern application, in which an argument P_2 is applied to an argument P_1. There is a single *reduction rule* called β-reduction:

$$(\lambda\alpha.P)Q \Rightarrow P[Q \to \alpha], \qquad (2)$$

where $P[Q \to \alpha]$ is a pattern obtained by substituting Q into α. Then the reduction in Fig. 2 can be written using term notations by:

$$(\lambda\alpha.S[G, G1]\alpha)(E[G1]) \Rightarrow S[G, G1]E[G1]. \qquad (3)$$

IV. Concluding Remarks

We introduced assurance carrying code, a framework for assuring the dependability of software supply chains. We are formalizing GSN and its patterns using λ calculus. Our progress will be presented in the near future.

Acknowledgement

The research was supported by ROIS NII Open Collaborative Research 2021-(21S0802).

References

[1] C. W. Johnson, "You outsource the service but not the risk: Supply chain risk management for the cyber security of safety critical systems," 2016.

[2] J. Boyens, C. Paulsen, R. Moorthy, N. Bartol, and S. A. Shankles, "Supply chain risk management practices for federal information systems and organizations," *NIST Special Publication*, vol. 800, no. 161, p. 32, 2015.

[3] JSOF, "Ripple 20, 19 Zero-Day Vulnerabilities Amplified by the Supply Chain," https://www.jsof-tech.com/disclosures/ripple20/, Accessed: July 25, 2021.

[4] U.S. Department of Homeland Security, "Emergency Directive 21-01," April 15, 2021.

[5] NIST CSRC, "NIST Releases Draft of NIST SP 800-161, Revision 1 for comment," May 10, 2021.

[6] T. Kelly, "Arguing safety – a systematic approach to safety case management," Ph.D. dissertation, Department of Computer Science, University of York, 1998.

[7] W. Howard, "The formulae-as-types notion of construction," *To H. B. Curry : Essays on combinatory logic, lambda-calculus, and formalism*, pp. 476–490, 1990.

[8] G. C. Necula, "Proof-carrying code," in *Proc. ACM POPL'97*. ACM Press, 1997, pp. 106–119.

[9] Y. Matsuno, "D-Case Communicator: A web based GSN editor for multiple stakeholders," in *SAFECOMP 2017 Workshops*, ser. LNCS, vol. 10489. Springer, 2017, pp. 64–69.

[10] E. Denney and G. Pai, "A formal basis for safety case patterns," in *SAFECOMP*, 2013, pp. 21–32.

[11] T. Kelly and J. McDermid, "Safety case construction and reuse using patterns," in *SAFECOMP*, 1997, pp. 55–69.

[12] R. Alexander, T. Kelly, Z. Kurd, and J. Mcdermid, "Safety Cases for Advanced Control Software : Safety Case Patterns," Tech. Rep., 2007.

[13] Y. Matsuno, "A Design and Implementation of an Assurance Case Language," *Proceedings - 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2014*, pp. 630–641, 2014.