A Reasonable Application of the Assurance Package CAP-B to Software Package Repositories

Hideaki Nishihara
National Institute of Advanced
Industrial Science and Technology (AIST)
Japan
h.nishihara@aist.go.jp

Yoriyuki Yamagata
University of Fukui
Japan
yoriyuki@u-fukui.ac.jp

Yutaka Matsuno
Nihon University
Japan
matsuno.yutaka@nihon-u.ac.jp

***Abstract*—Modern software systems are composed of various components, several of which are provided by third parties. This makes ensuring system security more challenging as the development information for these components is not publicly available. Common Criteria defines composed assurance packages (CAPs) to ensure the integrity of components in systems. However, it does not fully address this issue, since CAPs require both of public and non-public information to demonstrate the conformance. In this study, we analyze the structure of CAP-B, the intermediate-level instance of CAPs, and identify the specific requirements that pertain to the distribution package of the systems. Hence, our approach facilitates system security verification and assurance only with publicly available information. In particular, software supply chains, where numerous software products participate, are potential application domains of our results. To illustrate this, we examine the Arduino software ecosystem, which has an infrastructure for distributing software packages and associated documentation, and for supporting tests. We propose extending this infrastructure by incorporating security consideration and tests enhancing the overall resilience of the ecosystem.**

***Keywords*-Security assurance, Common Criteria, Composite systems, Goal Structuring Notation, Supply chain security, Arduino**

I. Introduction

Modern software systems are composed of various components. It realizes high productivity and flexibility in system development, particularly through the use of outsourced elements such as commercial off-the-shelf (COTS) products. However, this integration introduces challenges in fully understanding system processes and asserting their security. Verifying every functionality of a target system in detail is often impractical. Moreover, unexpected operational environments or unforeseen inputs can significantly impact the behavior of a system, potentially resulting in severe consequences.

Security assurance [1] addresses these challenges by offering evidence-based arguments to support security claims. These arguments are especially useful for establishing accountability, when the target system is complex and interacts with numerous other systems or stakeholders. Typically, security assurance is established by demonstrating conformance with security standards or guidelines. Software Bill-Of-Materials (SBOMs) [2] is being prepared recently as a litetweight security assurance mechanism. However, it focuses on traceability and transparency and does not show security of systems directly. One prominent standard is Common Criteria (CC) [3] which provides a comprehensive framework covering the structure of security concepts and the format of assurance documentation. In particular, CC defines the composed assurance packages (CAPs) for systems made up of multiple components (called composite systems), emphasizing the interactions and interdependencies among these components. These packages comprise lists of security requirements along with accompanying descriptions. However, the lack of explicitly stated connections among these requirements often makes the package difficult to comprehend.

Over the past few decades, formal approaches to build assurances have been developed, including the tool Goal Structuring Notation (GSN) [4]. GSN helps visualize assurance structures and support design and development of assurances. While the original domain that they are applied is safety domain as reviewd in Section II, the methodologies have spread to other domain such as security.

This paper analyzes and clarifies the structure of CAP-B, the intermediate-level instance of CAPs, with use of GSN. The CAP-B requirements are grouped based on the type of information and resources they depend on. The first group relates to the distribution of the target system, including the operational and installation manuals. The second group focuses on Security Target (ST), the assurance documentation specified in CC. The last group pertains to development information that is not always available to third parties or system users. This clarification offers a novel approach and is summarized to the GSN diagram. It enables the selective application of CAP-B requirements based on factors such as costs, available information, and specific system contexts.

We also explore the potential application of our findings within the open-source software (OSS) ecosystem. OSS development processes are often opaque, updates may be irregular, and much of the development lacks documentation. In addition, many OSS components rely on freely available reference components with minimal quality verification. While this model provides flexibility and high performance for developers, it poses challenges for security assurance, which typically requires extensive develompment documentation. In this study, we examine the Arduino software ecosystem [5], which is one of the major platform for designing and man-

ufacturing electronic devices and software, and evaluate the feasibility of providing assurance evidence for CAP-B. The Arduino ecosystem possesses the infrastructure to manage their products such as software repositories, conventions and policies for integrity, as well as testing frameworks. We can add security consideration aligned with CAP-B to the infrastructure, thereby participants in the ecosystem can contribute to or confirm the authorized security of its products cost-effectively.

Organization: The remainder of this paper is structured as follows. Section II introduces GSN as a foundational tool for system assurance. Section III summarizes the CC framework and CAP structure. Section IV presents a detailed analysis of CAP-B, with its structural representation (figured as in Fig. 4), which serves as the main result of this paper. Section V discusses practical applications of the findings. Section VI reviews the previous research on security assurances and standards, including supply chain security. Finally, concluding remarks are presented.

II. System Assurances and Goal Structuring Notation (GSN)

Assurance cases represent "a clear, comprehensive, and defensible argument that a system operates safely in a given context" [6], certifying safety-critical systems during development and operation. Assurance cases have been widely used to develop safety-critical systems ([7], [8]). Given the increasing importance of cybersecurity, security assurance cases have become a focal point and are actively being developed ([1], [9], [10]). While assurance cases are typically documented as texts, graphical notations have been proposed to clarify the argumentative structure of assurance cases. Notable examples include Goal Structuring Notation (GSN) [6] and Claim Argument Evidence (CAE) [11]. This paper uses GSN for documenting assurance cases.

Arguments in GSN are structured as trees with a few kinds of nodes, including: *goal* nodes for claims to be argued for, *strategy* nodes for reasoning steps that decompose a goal into sub goals, and *evidence* (solution) nodes for references to direct evidence that respective goals hold. Fig. 1 is a simple example of GSN. The root of the tree must be a goal node, called the top goal, which is the claim to be argued (G_1 in Fig. 1). For G_1, a context node C_1 is attached to complement G_1. Context nodes are used to describe the context (environment) of the goal attached to. A goal node is decomposed through a strategy node (S_1) into sub goal nodes (G_2 and G_3). The strategy node contains an explanation, or reason, for why the goal is achieved when the sub goals are achieved. S_1 explains the method used when arguing (arguments over each possible fault: A and B). When successive decompositions reach a sub goal (G_2) that has a direct evidence of success, an evidence node (E_1) referring to the evidence is added. Here we use a result of fault tree analysis (FTA) as the evidence. For the sub goal (G_3) that is not decomposed nor supported by evidence, a node (a diamond) of type *undeveloped* is attached to highlight the incomplete status of the case.

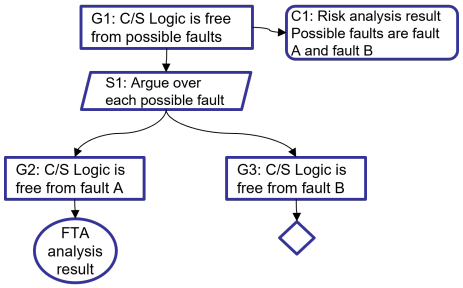


Fig. 1. An example of GSN diagrams

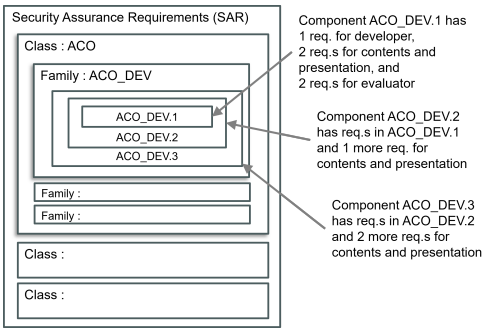


Fig. 2. Hierarchical structure of SAR

III. Composite Assurance Package in Common Criteria

A. Common Criteria

Common Criteria(CC) [3][1] is a primary international standard for the security evaluation of information systems. It specifies two types of security documentation for the target of evaluation (TOE): protection profiles (PP) and security targets (ST). It also includes a comprehensive set of security requirements that can be referred to by PPs and STs. Security assurance requirements (SAR) is a subset of these requirements, which is organized hierarchically (Fig. 2). SAR is structured into classes, each of which is further divided into families based on specific security aspects. Each family contains one or more components, which outline concrete requirements. These components are ordered by increasing levels of confidence. Every SAR requirement specifies either a developer action, a claim for the contents and presentation of TOE, or an evaluator action. For example, consider the SAR requirement:

> ACO_DEV.1.1D The developer shall provide development information for the base component.

This is the first requirement of the developer's actions (indicated by "1D" in the index) in the ACO_DEV.1 component. Here, the identifier ACO_DEV stands for "Development evidence" family (DEV) in "Composition" class (ACO).

A package in CC is a collection of security requirements grouped for specific purpose or scope. It consists of selected

[1]In the sequel, the reference "Common Criteria" and its abbreviation "CC" refer to the latest version CC2022.

requirements drawn from various parts of CC. Evaluation assurance levels (EALs) are well-known packages defined in CC. Each EAL provides a balanced set of security requirements corresponding to different assurance needs. For instance, EAL 7, the highest level, is typically required for highly security critical systems such as those involved in national defense and security.

B. Compositions in CC and CAP Package

Some sections of CC explicitly address the security evaluation of composed IT products, that are defined as systems compromising plural separately identified components. This emphasis on compositions is one of the unique features of CC. Chapter 14 of CC Part 1 discusses the assurance of composed products, referred to as composed TOEs from a general perspective. The key idea is to separate the assurances of individual components from those of the composed TOE. All components are assumed to have already undergone independent evaluation; therefore, the assurance process for the composed TOE focuses on the interactions among those components.

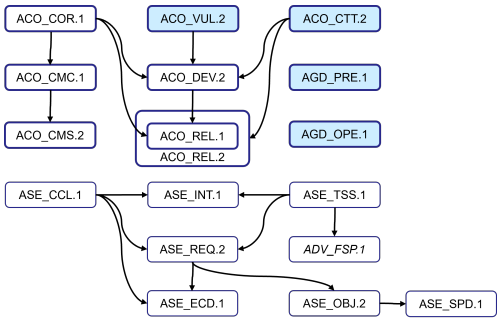
The primary composition pattern described in CC is the layered composition. In this model, a system consists of base components and dependent components. These components operate independently, while the dependent component utilizes the functionalities of the base components but not vice versa. Various aspects of composite TOEs[2] are analyzed and specified as requirements. ACO class in SAR addresses general compositons, while it actually focused on the layered model. Additionally, composition-related requirements are specified in other assurance classes, such as the ADV_COMP family in the ADV class.

Composed assurance packages (CAPs) are defined in CC Part 5 and explicitly addresses the assurances of composite TOEs. While CAPs do not cover the assurances of individual components, they focus on the interactions between base and dependent components, interfaces, reliance information, and configuration management. As CAPs, 3 packages are defined as follows. CAP-A is the basic package, which ensures the composite TOE is *structurally composed*, and which does not require involvement of developers of the base component. CAP-B is the middle-level package that supports that the composite TOE is *methodically composed*, and which requires partial development information from the developers of the base components. CAP-C is the highest level package which considers the inside of the base component additionally, meaning that it requires cooperations with the developers of the base component.

IV. Structural Analysis of CAP-B

CAP-B evaluates whether the composite TOE is *methodically composed*. In other words, beyond the scope of CAP-A, which focuses on ensuring consistent interactions among TOE components, CAP-B also examines the effects of those interactions. Although CAP-B offers a lower level of assurance than its superset CAP-C, it is more practical and easier to apply, as it requires only limited knowledge about the internal workings of the base components.

[2]This word is used for composed TOEs with layered composition models.



The blue-filled boxes belong to Group D in Table I. ADV_FSP.1 is not referred to in CAP-B but is included in this graph as ASE_TSS.1 depends on it.

Fig. 3. Dependency graph of CAP-B components:

Table I listss SAR components selected for CAP-B and Fig. 3 presents the dependency graph extracted from the descriptions of these assurance components. Notably, CAP-B directly references ACO_REL.1 whereas ACO_CTT.2, included in CAP-B, depends on its superset ACO_REL.2. Therefore, in this study, we analyze CAP-B in conjunction with ACO_REL.2.

First, we observe that the assurance components in Fig. 3 can be categorized into two distinct groups: components labeled with the prefix "ASE" are separate from the others. Additionally, Table I shows that certain components labeled "ACO" or "AGD" address aspects of the TOE that are accessible to users or third-parties. Therefore, we classify the assurance components in CAP-B into three groups. In Table I, the rightmost column shows the group to which each component belongs.

- Assurance components marked with D have requirements related to distributed materials. These components, including software packages and accompanying documents, are accessible to users of the TOE and can thus be verified directly.
- Assurance components marked with N have requirements concerning undisclosed development information. This includes detailed design documents, composition rationale, and configuration changes, that are not publicly available. Parties interested in reviewing this information must obtain permission from the developers.
- The rest of components in CAP-B (marked with S) pertain to the formal structure of ST, such as its organization and the information it includes.

This classification informs the CAP-B structure, which is illustrated by the GSN diagram in Fig. 4. The top goal G0 is "*Composite TOE is methodically composed*," which is the primary objective of CAP-B. This goal is decomposed into subgoals GS, GN, and GD, corresponding to the S, N, and D assurance components, respectively. Additionally, since ACO_COR.1 addresses the assurance of individual compo-

TABLE I
SAR COMPONENTS IN CAP-B PACKAGE

Assurance component	Subject and description	#Req.s in the comp. (∗)	Group
ACO_COR.1	**Composition rationale**: The demonstration that the base component can provide an appropriate level of assurance for use in composition.	1, 1, 1	N
ACO_CTT.2	**Rigorous interface testing**: Ensuring that each interface of the base component, on which the dependent component relies, is tested.	4, 4, 3	D
ACO_DEV.2	**Basic evidence of design**: Examination that the description of interfaces in the base component are consistent with the description of interfaces on which the dependent component relies.	1, 3, 2	N
ACO_REL.2	**Reliance information**: Provision of evidence that describes the reliance that a dependent component has upon the base component.	1, 4, 1	N
ACO_VUL.2	**Composition vulnerability analysis**: Analysis of vulnerability information available in the public domain and of vulnerabilities that may be introduced as a result of the composition.	1, 1, 5	D
AGD_OPE.1	**Operational user guidance**: Minimizing the risk of human or others errors in operation resulting in an undetected insecure state.	1, 7, 1	D
AGD_PRE.1	**Preparative procedures**: Encouraging secure transition from the delivered TOE to its initial operational environment.	1, 2, 2	D
ALC_CMC.1	**Labelling of the TOE**: Unique reference to ensure that there is no ambiguity in terms of which instance of the TOE is being evaluated.	1, 1, 1	N
ALC_CMS.2	**Parts of the TOE CM coverage**: Placing the TOE itself, its parts, and the evaluation evidence required by the other SARs under CM(configuration management) provides assurance that they have been modified in a controlled manner with proper authorizations.	1, 3, 1	N
ASE_CCL.1	**Conformance claims**: Determining the validity of the conformance claim.	2, 13, 1	S
ASE_ECD.1	**Extended components definition**: Evaluation of the definition of extended components.	2, 5, 2	S
ASE_INT.1	**ST introduction**: Description of the TOE in a narrative way on three levels of abstraction.	1, 9, 2	S
ASE_OBJ.2	**Security objectives**: Demonstrating that the security objectives adequately and completely address the security problem definition.	2, 3, 1	S
ASE_REQ.2	**Stated security requirements**: Evaluation of the security requirements to ensure that they are clear, unambiguous and well-defined.	2, 12, 1	S
ASE_SPD.1	**Security problem definition**: Definition of the security problem to be addressed by the TOE and the operational environment of the TOE.	1,4,1	S
ASE_TSS.1	**TOE summary specification**: Evaluation of the TOE summary specification to determine whether it is adequately described how the TOE meets it SFRs, and protects itself against interference, logical tampering and bypass.	1, 1, 2	S

(∗)Three numbers count requirements for developers, for contents and presentation, and for evaluators in order.

nents, we introduce the fourth goal GN' to represent these requirements. Dependencies on N assurance components are indicated using context nodes.

Subgoals GN and GD are further decomposed into specific goals. For this analysis, we assume the TOE includes only one base component. While CAPs generally accommodate TOEs with multiple base components supporting a single dependent component, we focus this simpler case. The arguments on the goal GS (related to the ST) and GN' (related to individual system components) are left undeveloped due to the space constraints.

The lowest-level goals (labeled GN1, GN2, etc.) are supported by evidence. Each requirement in CAP-B (listed in Table II) is linked to the corresponding evidence in Fig. 4. In other words, each requirement is satisfied through specific evidence. Moreover, it also shows that each assurance component in CAP-B contributes a single goal except for ACO_COR.1.

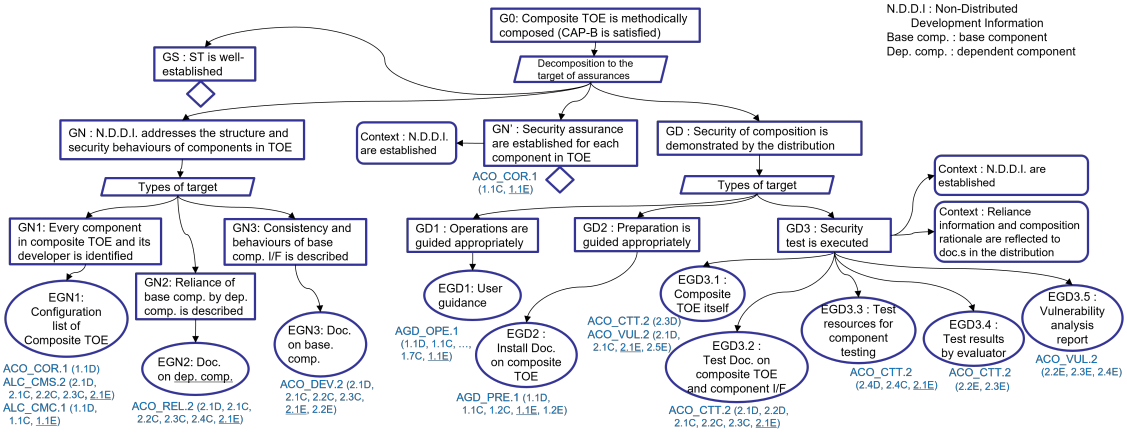
We also highlight dependencies between assurance components. As shown in Fig. 3, D-marked components ACO_CTT.2 and ACO_VUL.2 depend on N-marked components which it influence the assurance of the goal GD3. However, at the requirement level, only ACO_VUL.2.4E, concerning vulnerability analysis, explicitly refers to undisclosed development information. Accordingly, two context nodes are added in Fig. 4 to reflect this relationship.

We evaluate our result by comparing with the previous research [12], which analyzes CC 3.1 (Revision 3) [13] and represents the structure of the ATE class in SAR as a GSN diagram. ATE addresses testing and thus the top goal of the GSN diagram in that study reflects the objective of ATE. The entire structure of the GSN diagaram follows the hierarchical structure of ATE: each subgoal of the top goal corresponds to a family in ATE, and it is decomposed to subgoals corresponding to assurance components in the family as well. The lowest-level goals are individual requirements in ATE, and they are linked to possible evidence for them. Our result is comparable to that study in a couple of aspects and has advantages. As trees, the GSN diagram in that study has depth 6 and so has our result (see Fig. 4). Both diagrams are not so complicated to trace arguments from every evidence to the top goal. Our analysis is based on the available information of TOE, and does not strictly adhere to the hierarchical structures of SAR. It allows us to select reasonable requirements in CC and apply them for various styles of system development. In our result, individual requirements in CAP-B are only linked to the corresponding evidence. It is not formal compared to the diagram in [12], but it contributes to simplifying the assurance structure. There are so many requirements in assurance families in SAR and it increases the volume of the assurances if every requirement is explicitly addressed, in particular, in graphical presentations.

V. APPLICATION

A. Idea

Modern software development increasingly relies on third-party software components. Developers frequently source software packages from repositories and integrate them to build



Requirement identifiers from CAP-B are shown alongside the relevant evidence. In particular, the specific requirements that ask evaluators to verify the provided information are underlined.

Fig. 4. A GSN diagram for CAP-B

their products. These resulting products are then packaged and may, in turn, be used as components in larger software systems. In such ecosystems, quality and security challenges occur frequently as reported in [14]. While details of development must be examined to construct assurances for software packages, much of development information, particularly concerning security, is not shared. Moreover, the development processes are either inadequately addressed or undisclosed. Therefore, preparing security assurances of a software package by CC substantially increases efforts especially for non-proprietary software.

The analysis of CAP-B in the previous section offers a potential solution for establishing reasonable security assurances. Typically, the materials in the distribution of a software package include the software itself, along with several documents that describe its interfaces, specifications, operations, and other relevant details. These materials have room for evidence that supports the subgoals of the goal GD in Fig. 4. In other words, by the distribution augmented with security-related information, the claim "*Security of composition is demonstrated by the distribution*" is ensured. Although the goal GD is a subgoal that supports the top goal G0, it explicitly addresses all CAP-B requirements related to distributed materials (enumerated in Table II). Thus, in this context, the TOE can be considered compliant with CAP-B in specific aspects.

Additionally, if the developer of the TOE takes a cost and provides documented development information, such as composition configuration, reliance on the base components, and detailed functionality of the base components, then they serve as the evidence and users of the TOE can confirm the goal GN. In addition to the assurance of GD discussed above, this further reinforces ensuring the methodical composition of the TOE, the top goal G0. Furthermore, preparing ST for the TOE ensures the goal GS, thereby demonstrating the TOE is fully aligned with CAP-B, together with the security augmented distribution package and documentations on develeopment information,

B. Addition of security information to software packages

As a case study, we consider Arduino [5], a platform used for designing and manufacturing electronic devices and software. Arduino systems can be integrated with devices, such as communication boards, cameras, sensors, and motors.

To enhance productivity and control capabilities, a software library repository [15] is provided for Arduino systems. One such library, ArduinoCore-API [16], defines abstract classes and methods for interfacing with external devices. It is distributed with instruction documents as attached files, and with test codes (based on Catch 2 framework [17]) for several classes. Therefore, by inheriting from ArduinoCore-API, control software for individual external devices is implemented. It allows Arduino application programs (referred to as "sketches") to interact with the devices in a standardized manner (see Fig. 5 and the example below). Moreover, users and third-parties can examine the quality of the control software, if the package distribution also inherits above auxiliary materials from the ArduinoCore-API package.

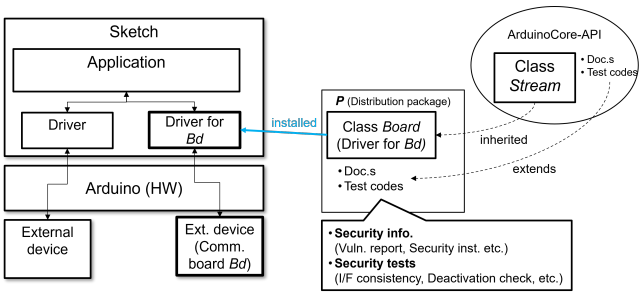


Fig. 5. A control software for Arduino system inheriting from an ArduinoCore-API class

In our context, introducing security aspects to the software package could give the security assurance of the software with minimal additional effort. As an example, we consider a library package *P* released for a communication board *Bd*.

TABLE II
REQUIREMENTS AND CORRESPONDING EVIDENCE UNDER THE GOAL GD IN FIG. 4

Identifier	**Requirement** (simplified)	**Evidence**
ACO_CTT.2.1D	Providing composed TOE test doc.s	Test doc.s or comments in test codes for composed TOE
ACO_CTT.2.2D	Providing base component I/F test doc.s	Test doc.s or comments in test codes for the base component
ACO_CTT.2.3D	Providing the composed TOE for testing	TOE itself
ACO_CTT.2.4D	Providing an equivalent set of resources for developer's functional testing of the base component	Test documentations for composed TOE
ACO_CTT.2.1C	Composed TOE and the base component I/F test doc.s, consisting of test plans, expected test results and actual test results	Test doc.s, test codes, and comments in them
ACO_CTT.2.2C	Test doc.s from the developer executions of the base component I/F tests, demonstrating that TSF behaves as specified and complete	Test doc.s, test codes, and comments in them
ACO_CTT.2.3C	Test doc.s from the developer executions of the base component I/F tests, demonstrating that the base component I/F relied upon by the dependent component behaves as specified and complete	Test doc.s, test codes, and comments in them for the base component
ACO_CTT.2.4C	The base component suitable for testing	Notes for access to the base component
ACO_CTT.2.2E	Executing a sample of test in the test doc.s	Executable test codes or reports on the test execution
ACO_CTT.2.3E	Testing subset of the TSF I/F of composed TOE to confirm that the composed TSF operates as specified	Executable test codes or report on the test execution and analysis
ACO_VUL.2.1D	Providing the composed TOE for testing	TOE itself
ACO_VUL.2.1C	Composed TOE suitable for testing	TOE itself
ACO_VUL.2.2E	Performing analysis to determine that any residual vulnerabilities identified for the base and dependent component are not exploitable in the operational environment	Vulnerability report
ACO_VUL.2.3E	Performing a search of public domain sources to identify possible vulnerabilities in the composed TOE operational environment	Vulnerability report
ACO_VUL.2.4E	Performing an independent vulnerability analysis of the composed TOE, using guidance documentation, reliance information, and composition rationale.	Vulnerability report
ACO_VUL.2.5E	Execution of penetration testing, to demonstrate that the composed TOE is resistant to attacks with basic attack potential	Penetration test report
AGD_OPE.1.1D	Providing operational user guidance	Operational user guidance
AGD_OPE.1.1C	Operational user guidance describing user-accessible functions and privileges	Corresponding descriptions in operational user guidance
AGD_OPE.1.2C	Operational user guidance describing how to use the available I/F in a secure manner	Corresponding descriptions in operational user guidance
AGD_OPE.1.3C	Operational user guidance describing the available functions and I/F	Corresponding descriptions in operational user guidance
AGD_OPE.1.4C	Operational user guidance presenting each type of security-relevant event relative to user-accessible functions	Corresponding descriptions in operational user guidance
AGD_OPE.1.5C	Operational user guidance identifying all possible modes of operation of TOE, consequences, and implications for maintenance	Corresponding descriptions in operational user guidance
AGD_OPE.1.6C	Operational user guidance describing the security controls	Corresponding descriptions in operational user guidance
AGD_OPE.1.7C	Clear and reasonable operational user guidance	Operational user guidance
AGD_PRE.1.1D	Providing TOE including preparative procedures	TOE and the preparation document
AGD_PRE.1.1C	Preparative procedure describing all the steps necessary for secure acceptance of the delivered TOE	Preparation document
AGD_PRE.1.2C	Preparative procedure describing all the steps necessary for secure installation of the TOE and secure preparation of the operational environment	Preparation document
AGD_PRE.1.2E	Application of Preparative procedure to confirm that TOE can be prepared securely for operation	Execution report

NOTE 1: I/F and TSF are abbreviations of interface and TOE security functionality respectively.
NOTE 2: ACO_CTT.2.1E, ACO_VUL2.1E, AGD_OPE.1.1E, and AGD_PRE.1.1E are omitted as they simply require to confirm other requirements in the assurance component they belong to are satisfied.

The library *P* inherits from the *Stream* class defined in the ArduinoCore-API library. Specifically, the APIs declared in the *Stream* class are implemented in *P*, and so are functionalities specific to the board *Bd*. The package *P* includes the library itself, along with related documentations and test codes for it (they reference materials in the *Stream* class as mentioned above). This configuration enables us to treat *Bd* and *P* as a composite TOE, where they serve as the base component and the dependent component, respectively. Security assurance information can be incorporated into the distributed materials to confirm the goal GD in Fig. 4 by addressing all its subgoals as follows:

a) GD1: "Operations are guided appropriately": This goal is supported by TOE user guidance that fulfills the requirements in AGD_OPE.1. The guidance for *P* builds upon the documentation for the *Stream* class and includes operational instructions tailored to *P* and *Bd*. As a part of security notes, we can address security concerns on *P* and *Bd*:

- User-accessible functions and associated privileges (as per AGD_OPE.1.1C),
- Secure usage of available interfaces (as per AGD_OPE.1.2C),
- Security-relevant events (as per AGD_OPE.1.4C).

The resulting information constitutes the evidence EGD1, supporting the goal GD1.

b) GD2: "Preparation is guided appropriately": Detailed installation procedure for setting up *P* with *Bd* securely can be included in the installation documentation, as required by AGD_PRE.1.

c) GD3: "Security test is executed": The evidence EGD3.1 is provided through the distributed materials that inevitably include the composite TOE (*P* and *Bd*). Observe that the test codes bundled with the *Stream* class verify the general functionalities of APIs, including:

- Successful reading from virtual input streams (as the library is hardware-independent and test suites cannot assume any physical data streams).
- Correct formatting and interpretation of virtual input data.
- Proper operation of lookup functions on virtual inputs.
- Correct setting and retrieval of parameters to/from the device.

The library *P* inherits from the *Stream* class, and its test codes reflect this inheritance. We prepare for the security test codes for *P*, with Catch 2 framework, in addition to functional test suites and distribute them in one package. The security tests verify:

- Interface consistency between *P* and *Bd* (as per ACO_CTT.2.1C and ACO_CTT.2.3C), where functional test cases for *P* may take their place,
- Rejection of unsuitable parameter values in communication protocols (as per ACO_CTT.2.2C),
- Prevention of invoking unused device functionalities in Arduino applications (as per ACO_CTT.2.2C).

The code fragment in Fig. 6 shows a possible implementation of them. The test in the first WHEN clause checks whether *P* and *Bd* work together in a mode without any problem. Remark that the supplementary tests are recommended in the comment. The tests in the second WHEN clause verify whether irregular parameters or prevented function requests are refused by *P*.

```
// Identifiers starting with '_' mean they are constants
// defined in advance.

TEST_CASE (
  "Check for inadequate operations",
  "[Security-ACO_CTT.2]")
{
  Board bd( _port, _bps, _mode);  // Initialization
  // Board class is defined in the package P

  WHEN ("Check the transmission capability")
  {
    res =bd.send( _specific_test_data);
    REQUIRE(res == TRUE);
    // P processed the test data without problem.
    // The corresponding output from Bd must be verified.
  }
  WHEN ("Attempt to change to invalid mode.")
  {
    REQUIRE(bd.changemode( _non_existent_id) == _error);
    REQUIRE(bd.changemode( _forbidden_mode_id) == _error);
    // No backdoors are left.
  }
}
```

Fig. 6. A fragment of the additional security test suite for *P*

These test codes, along with associated comments and documentation, constitute the evidence EGD3.2. They also fulfill EGD3.3, when accompanied by details of the testing environment.

The evidence EGD3.4 is derived from evaluator activities. In OSS context, any participants including users and third-parties can commit to security of products. Therefore, they can take the role of evaluators; they review the evidence EGD3.1, EGD3.2, and EGD3.3, verify *P* with given test codes, and report the result as EGD3.4.

The evidence EGD3.5 addresses known vulnerabilities in the TOE. They are investigated by the evaluators, who are participants of the ecosystem. The developers of *P* can assist by referencing the public vulnerability databases or documenting security considerations made during development.

In summary, all evidence supporting the goal GD is derived solely from the distribution package of *P* and publicly available information. This confirms that the package is methodically composed, thereby enhancing its security posture.

VI. Related Works

Assurance structures are typically represented as trees; therefore, decomposing assurance claims is a key challenge in constructing assurance cases. In [18], a method for claim decomposition is proposed based on five core security attributes: confidentiality, integrity, availability, authentication and accountability. The article [12], with which we compare our result in Section IV, proposes decomposition of goals following the hierarchical structure of CC requrements.

Supply chain security has emerged as a major topic over the past decade. With the growing complexity and length of supply chains, tracing the history of products and components has become increasingly complex. If foundational components are compromised and remain undetected, then they may be repeatedly procured and integrated into various systems. Once it is revealed after market release, numerous systems in operation are known vulnerable and remarkably large effort is required to identify and treat the risk. Several guidelines [19], [20], [21] on supply chain security have been published addressing organizational, technical, and procedural aspects in a integrated manner. In [22], supply chain risk management is examined at the governance level with security cases presented as illustrative examples.

One notable area of focus is the software Bill of materials (SBOMs) which has garnerd substantial interest from the industry. SBOMs provide detailed product metadata, including authorship, configurations, and dependency information, as specified in [2], [23]. While SBOMs offer benefits such as machine-readability and components traceability, they primarily function as metadata inventories. In contrast, assurance cases demonstrate arguments directly and provide confidence about the security of systems.

Demonstration of conformance to technical standards is one of main purposes of assurance cases. To conduct it effectively, technical standards themselves have been analyzed in literature. The article [24] offers interpretations of three standards on information security [25], avionics safety [26], and risk management of medical devices [27]. The proposed method extracts key characteristics of these standards and identifies the challenge involved in building assurance cases based on them.

In [12] (reviewed in Section IV as mentioned), a method is proposed to evaluate technical standards specifically from the perspective of assurance cases development. Furthermore, the article [28] discusses process patterns in Common Criteria 3.1 (Revision 4), particularly, focusing on how system developers and evaluation facilities collaborate in the certification process. In [29], a set of principles is proposed for developing assurance case templates, which involves modeling technical standards to serve as a formulation for assurance cases and considering how they apply across product lines.

VII. Concluding Remarks

In this research, we investigate a Composite Assurance Package B (CAP-B) defined in Common Criteria. Requirements in CAP-B are addressed with distinct aspects of TOE, and hence they are classified into three groups. It implies a structure of CAP-B, in particular, reasonable assurances based on the distributed materials are distinguished. We applied the result to software package ecosystems. Modern software package repositories already offer resources to claim the software quality of individual packages. Adding adequate security information or security test codes to the package satisfies CAP-B requirements with reasonable costs.

The idea of SBOM also seems security assurances with reasonable costs. Indeed, some entries described in SBOM can be linked to requirements in CAP-B. However, they are indirect; they help to trace the composition of system components but does not give explicit rationale for security. The aspects CAP-B covers are limited and the holistic security assurances for information systems would need arguments from the various viewpoints. The feasible combinations of standards, guidelines, or principles, including SBOMs, should be established to improve the security. In particular, tool supports that derive both of SBOM entries and evidence for CAP-B assurances from development information will contribute it.

In actual system development, many components are involved and the GSN diagrams instantiated from Fig. 4 becomes complicated. Hence, a new framework is expected to design and build assurances and GSN presentations systematically. It allows modifying or synthesizing fragments of GSN diagrams and makes a complete diagram in formal way. We can prepare an assurance template and apply it with the framework, which will accelerate assuring security of information systems effectively.

Acknowledgement

This research was supported by JSPS KAKENHI Grant Number JP23H03376.

References

[1] M. Mohamad, J. Steghöfer, and R. Scandariato, "Security assurance cases - state of the art of an emerging approach," *Empir. Softw. Eng.*, vol. 26, no. 4, p. 70, 2021.

[2] The United States Department of Commerce, "The Minimum Elements For a Software Bill of Materials (SBOM)," 2021.

[3] CCRA, "Common Criteria for Information Technology Security Evaluation Rev. 1 (CC:2022)," 2022.

[4] Assurance Case Working Group, "Goal structuring notation community standard version 3," May 2021, downloaded from https://scsc.uk/r141C:1?t=1.

[5] "About Arduino," (Version: Sept. 15 2021). [Online]. Available: https://www.arduino.cc/en/about

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

[7] European Organisation for the Safety of Air Navigation, "Safety case development manual," 2006.

[8] R. Hawkins, T. Kelly, and I. Habli, "Developing assurance cases for D-MILS systems," in *International Workshop on MILS: Architecture and Assurance for Secure Systems, MILS@HiPEAC 2015*, 2015.

[9] A. Shukla et al., "System security assurance: A systematic literature review," *Comput. Sci. Rev.*, vol. 45, p. 100496, 2022.

[10] ISO TR 15443-1:2012, "Information technology – Security techniques – Security assurance framework – Part 1: Introduction and concepts," 2012.

[11] R. Bloomfield and P. Bishop, "Safety and assurance cases: Past, present and possible future - an Adelard perspective," in *Proceedings of the Eighteenth Safety-Critical Systems Symposium*, 2010.

[12] P. J. Graydon and T. P. Kelly, "Using argumentation to evaluate software assurance standards," *Inf. Softw. Technol.*, vol. 55, no. 9, pp. 1551–1562, 2013.

[13] CCRA, "Common Criteria for Information Technology Security Evaluation Version 3.1 Revision 3," 2009.

[14] S. Zajdel, D. E. Costa, and H. Mili, "Open source software: an approach to controlling usage and risk in application ecosystems," in *Proceedings of the 26th ACM International Systems and Software Product Line Conference (SPLC '22)*, 2022.

[15] "Libraries for Arduino." [Online]. Available: https://docs.arduino.cc/libraries/

[16] "ArduinoCore-API," (Version: Release 1.5.1). [Online]. Available: https://github.com/arduino/ArduinoCore-API

[17] "Catch2 Project," (Version: July 3 2025). [Online]. Available: https://github.com/catchorg/Catch2

[18] J. Jaskolka, A. Jawad, J. Samuel, and B. Hamid, "A security property decomposition argument pattern for structured assurance case models," in *EuroPLoP'21: European Conference on Pattern Languages of Programs 2021*, 2021, pp. 24:1–24:10.

[19] NIST SP 800-161, "Cybersecurity Supply Chain Risk Management Practices for Systems and Organizations (Rev. 1)," 2024.

[20] NIST IR 7622, "Notional Supply Chain Risk Management Practices for Federal Information Systems," 2012.

[21] Cybersecurity and Infrastructure Security Agency, Federal Bureau of Investigation, and Australian Signals Directorate's Australian Cyber Security Centre, "Safe software deployment: How software manufacturers can ensure reliability for customers," October 2024.

[22] C. W. Johnson, "You Outsource the Service but Not the Risk: Supply Chain Risk Management for the Cyber Security of Safety Critical Systems," in *34th International System Safety Conference*, 2016. [Online]. Available: http://www.dcs.gla.ac.uk/ johnson/papers/ISSC16/supply.pdf

[23] ISO/IEC 5962:2021, "Information technology – SPDX® Specification V2.2.1," 2021.

[24] T. S. Ankrum and A. H. Kromholz, "Structured assurance cases: Three common standards," in *Ninth IEEE International Symposium on High Assurance Systems Engineering (HASE 2005)*, 2005, pp. 99–108.

[25] ISO 15408:1999, "Common Criteria for Information Technology Security Evaluation Version 2.1," 1999.

[26] RTCA DO-178B, "Software Considerations in Airborne Systems and Equipment Certification," 1992.

[27] ISO 14971:2000, "Medical devices – Application of risk management to medical devices," 2000.

[28] A. D. Sinnhofer, W. Raschke, C. Steger, and C. Kreiner, "Patterns for common criteria certification," in *Proceedings of the 20th European Conference on Pattern Languages of Programs, EuroPLoP 2015*, 2015, pp. 33:1–33:15.

[29] T. Chowdhury, C. Lin, B. Kim, M. Lawford, S. Shiraishi, and A. Wassyng, "Principles for systematic development of an assurance case template from ISO 26262," in *2017 IEEE International Symposium on Software Reliability Engineering Workshops, ISSRE Workshops*, 2017, pp. 69–72.