Evaluating Security Assurance Case Adaptation

Sharmin Jahan
University of Tulsa
shj594@utulsa.edu

Allen Marshall
University of Tulsa
allen-marshall@utulsa.edu

Rose F. Gamble
University of Tulsa
gamble@utulsa.edu

Abstract

Security certification processes for information systems involve expressing security controls as functional and non-functional requirements, monitoring deployed mechanisms that satisfy the requirements, and measuring the degree of confidence in system compliance. With the potential for systems to perform runtime self-adaptation, functional changes to remedy system performance may impact security control compliance. This impact can extend throughout a network of related controls causing significant degradation to the system's overall compliance status. We represent security controls as security assurance cases and implement them in XML for management and evaluation. The approach maps security controls to softgoals, introducing achievement weights to the assurance case structure as the foundation for determining security softgoal satisficing levels. Potential adaptations adjust the achievement weights to produce different satisficing levels. We show how the levels can be propagated within the network of related controls to assess the overall security control compliance of a potential adaptation.

1. Introduction

With the emergence of autonomous systems, multiple domains seek to exploit the potential benefits of allowing systems to make dynamic decisions and repairs at runtime. These systems may craft adaptations from internal resources [2, 4], from an external source like a cloud [23], or from machine learning or genetic algorithms [14] applied to existing code or historical execution traces. Security-critical systems need additional assurances that self-adaptation will not compromise their compliance with security constraints.

Assurance cases represent structured claims and arguments for those claims, crafted in the form of subclaims, strategies for satisfying the arguments, and evidence related to the strategies [8]. Assurance cases [22] are a tool for documenting how a system satisfies its requirements. An assurance case contains a root claim or goal stating a requirement, along with an argument indicating why the claim of compliance with the requirement should be trusted. An assurance case argument, structured as a tree, links the goal to subgoals and ultimately traces each subgoal to supporting evidence. Assurance cases are commonly represented using a graphical notation called Goal Structuring Notation (GSN) [11]. Security assurance cases have been used to focus on the acceptability of a security solution to known vulnerabilities. They may be domain-specific, given that certain types of systems and domains have known threat vectors. The overall objective is to determine the evidence to support threat mitigation [16].

Security controls are part of the compliance requirements for US federal information systems found in the NIST SP800-53 [19]. The 800-53 also outlines how organizations determine which security controls are relevant to their systems. Subsets of these controls have been extended to multiple domains. For example, any nonfederal business receiving Controlled Unclassified Information from a US government entity must show that they have a process in place to comply with the NIST SP800-171 [21], which contains a subset of the 800-53. Thus, understanding security controls and how to certify compliance is becoming a normal part of doing business with the US government. Certification requires that there are mechanisms deployed to ensure security control effectiveness for those controls necessary for secure system operation. Some level of risk is acceptable as part of the certification process, leading to different levels of system trustworthiness. Assurance cases for security controls should allow for the representation of these trustworthiness levels.

Information systems that can self-adapt their functionality, communication, and decision-making processes during runtime increase the burden of determining compliance with security requirements. The challenge is to determine how a functional change, not necessarily made for security reasons, can impact security control certification, and provide additional reasoning on whether to proceed with an adaptation.

In this paper, we express 800-53 security controls as functional and non-functional requirements by extending an initial security assurance case template [17]. We use the extended template to determine how a

URI: https://hdl.handle.net/10125/60168
ISBN: 978-0-9981331-2-6

functional adaptation that locally affects one security control can propagate to other security controls, reducing overall compliance confidence. The new template incorporates metadata from the security controls that detail what the control provides to and requires from predefined related controls, forming a network of dependencies among the controls.

We use the concept of network vulnerability metrics [24] to calculate achievement weights of subclaims (subgoals) that support each security assurance case claim (root goal). By expressing the security assurance case claims as softgoals [5], we adapt an existing algorithm [12] to measure the satisficing level of each security control and its impact on its related security controls in the network based on its local achievement weight. Because we allow adaptations to be configured at runtime, the change operation that the system performs may cause a reduction in the achievement weights associated with the subgoals, requiring a recalculation of the satisficing level at the root goal and throughout the network. We demonstrate the approach with a sample application, showing how the achievement weights and satisficing levels compare to the implementation of each potential adaptation defined.

2. Related Work

An assurance case evolution technique has been proposed using a model management approach [13]. with the objective of maximizing the reuse of assurance case components when evolution invalidates part of an assurance case. This framework provides an algorithm that identifies reusable components of an assurance case and relies on human intervention to correct branches that are not found to be reusable. Other exploration in evolving assurance cases considers the problem of checking the quality of a new assurance case that replaces a faulty assurance case once a flaw is detected [6, 7]. The approach represents flaws in the original assurance case using a formal problem model and can determine which of the problems are resolved by the new assurance case. Our work differs in its approach and several of its assumptions. We do not assume the availability of a set of flaw descriptions for the original assurance case, but instead assume that an impact assessment of the potential adaptations is provided that can be used to dynamically calculate achievement weights of the affected subgoals.

Assurance case quality evaluation has also been studied by Lin et al. [15]. Their approach assigns confidence levels to claims in assurance cases. It relies on the existence of a known, valid assurance case, and computes confidence levels for other assurance cases based on both metrics of similarity to the valid assurance case and the Dempster-Shafer theory for uncertain reasoning. We similarly assign numeric weights to main goals in the form of satisficing levels, and our approach also propagates achievement weights from child goals to parent goals. In contrast, we consider specific issues associated with security assurance cases for NIST security controls, such as the existence of control enhancements and related controls that propagate and affect networked compliance scores.

Lipson [16] states that the credibility of an assurance case depends on incorporating appropriate evidence into the argumentation. Failure to organize appropriate evidence may weaken the argument, because at a low level the argument is linked with its evidence. Organizing the evidence involves an understanding of claims (compliance goals), system context, system enablers (dependencies) and potential threats. The approach relies on a framework [8] to categorize security property evidence for the argument and a template to help evaluate the quality of evidence.

Security certification can be a useful technique for maintaining stable security behavior, because certification schemes involve evaluation of the system by considering security claims and evidence. A security certification framework for a cloud-based system is proposed to verify security certificate validation [1, 2]. In this framework, a certification model template and instance are developed, and probes are deployed to collect evidence about system consistency. Inconsistencies are listed, and adaptations can be triggered based on the misconfiguration report.

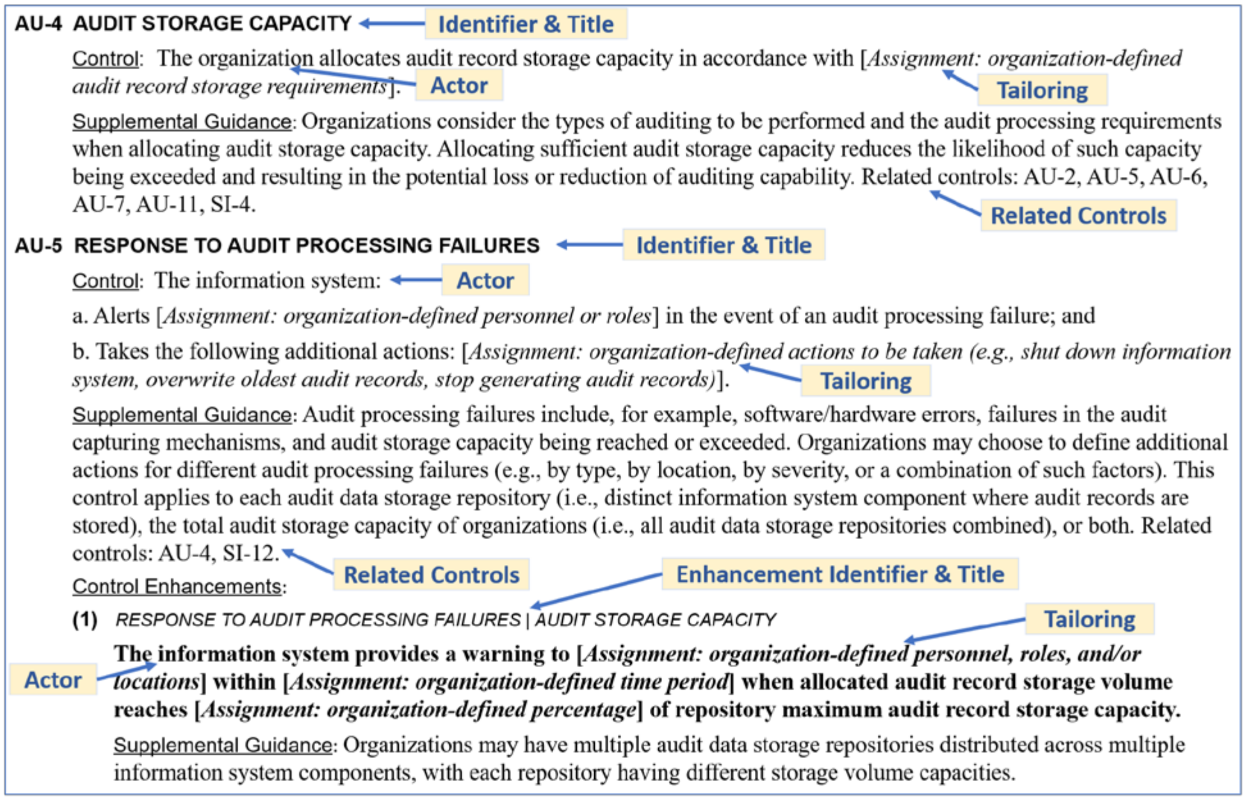
Chung and Nixon [5] designed a non-functional requirement (NFR) expression framework, in which they represent NFRs as "softgoals" and use the concept of "satisficing" to measure softgoal achievement. Satisficing provides a degree of satisfaction for a softgoal based on positive or negative evidence. It has been used to bridge nonfunctional and functional requirements by analyzing the interdependencies between them and provide insights for conflict analysis [18]. The framework defines a Softgoal Interdependency Graph (SIG) by decomposing the softgoals into subgoals based on AND/OR relationships over which a satisficing algorithm is performed. To quantitatively evaluate the SIG, a process called "Softgoal using Weight" (SGW) is deployed [12]. The SIG leaves are operational subgoals that assigned achievement weights by a subject matter expert based on the contribution value of the subgoal attribute to the parent goal. A satisficing algorithm uses the propagation of the achievement weights to quantify the impact the attributes have on the high-level goal.

Satisficing would be more applicable to a self-adaptive system if achievement weights could be calculated at runtime. One approach is to determine a quantitative *vulnerability metric* based on a defined community structure of a complex weighted network [24]. The *community* is defined as subnetwork of several nodes connected with a high degree of strength. Using hierarchical agglomerative algorithm (HAA), a community can be detected, along with its connections with other communities, to quantify vulnerability. The vulnerability metric depends on number and strength of external connections with other communities, the degree of dependency on other communities, and the internal (within a community) connection density and strength. The internal strength serves as primary factor for measuring vulnerability, which denotes the impact of the connection's weight on the community. It is this measure that can be used for an achievement weight.

3. NIST Security Controls

For the security assurance cases, we use the NIST SP800-53 [19] (called 800-53) security controls as shown in Figure 1. A security control associates a title with each identifier. AU-4 refers to the 4th control within the Audit family of controls. The actor is either the information system or the organization. The control statement follows the actor designation. It may be a single statement, like AU-4, or separated into distinct parts, like AU-5(a) and AU-5(b). The statement can contain a mix of functional and non-functional requirements. Tailoring, a major part of security control certification, is performed when the organization instantiates what is required by the [*Assignment*: …] for the information system under consideration.

The related controls infer a dependency relationship among the controls. For AU-5, they are AU-4 and SI-12. There are other controls that tag AU-5 as a related control, such as AU-4, with different dependencies. The relationships may be tightly coupled, where AU-5 relies on the audit storage capacity determined in AU-4, or loosely coupled, where AU-4 provides AU-5 with a parameter it obtains from its related control AU-11. These inter-dependencies can be used to assess the impact of a self-adaptation on not just a single security control, but on the network of security controls. AU-5(1) is a control enhancement, which provides additional specification decisions and constraints. The related controls can be inherited from the main control or the enhancement can have its own related controls that are not shared with the main control. Controls are assigned to a baseline set related to the impact on the confidentiality, integrity, or availability of the system if a breach occurs. For example, AU-5 appears in the baseline set for moderate impact, while AU-5(1) appears in the baseline set for high impact systems.



AU-4 AUDIT STORAGE CAPACITY

Control: The organization allocates audit record storage capacity in accordance with [*Assignment: organization-defined audit record storage requirements*].

Supplemental Guidance: Organizations consider the types of auditing to be performed and the audit processing requirements when allocating audit storage capacity. Allocating sufficient audit storage capacity reduces the likelihood of such capacity being exceeded and resulting in the potential loss or reduction of auditing capability. Related controls: AU-2, AU-5, AU-6, AU-7, AU-11, SI-4.

AU-5 RESPONSE TO AUDIT PROCESSING FAILURES

Control: The information system:

a. Alerts [*Assignment: organization-defined personnel or roles*] in the event of an audit processing failure; and

b. Takes the following additional actions: [*Assignment: organization-defined actions to be taken (e.g., shut down information system, overwrite oldest audit records, stop generating audit records)*].

Supplemental Guidance: Audit processing failures include, for example, software/hardware errors, failures in the audit capturing mechanisms, and audit storage capacity being reached or exceeded. Organizations may choose to define additional actions for different audit processing failures (e.g., by type, by location, by severity, or a combination of such factors). This control applies to each audit data storage repository (i.e., distinct information system component where audit records are stored), the total audit storage capacity of organizations (i.e., all audit data storage repositories combined), or both. Related controls: AU-4, SI-12.

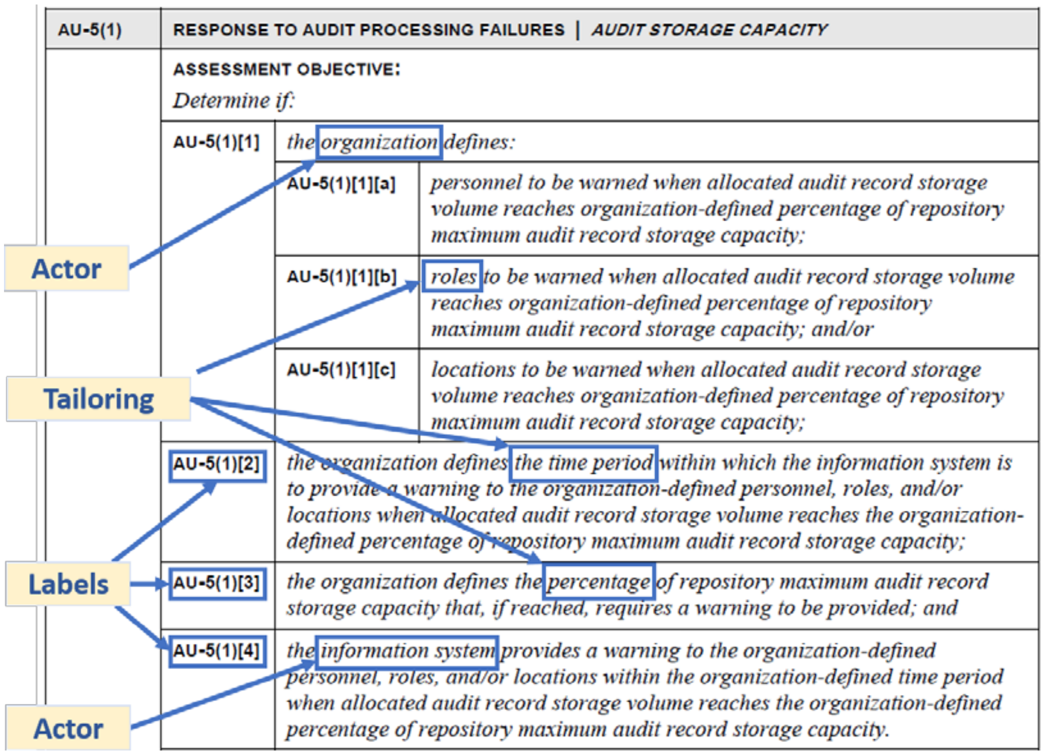
Control Enhancements:

(1) *RESPONSE TO AUDIT PROCESSING FAILURES | AUDIT STORAGE CAPACITY*

The information system provides a warning to [*Assignment: organization-defined personnel, roles, and/or locations*] within [*Assignment: organization-defined time period*] when allocated audit record storage volume reaches [*Assignment: organization-defined percentage*] of repository maximum audit record storage capacity.

Supplemental Guidance: Organizations may have multiple audit data storage repositories distributed across multiple information system components, with each repository having different storage volume capacities.

Figure 1. Security Controls AU-4, AU-5, and AU-5(1)


Actor
Tailoring
Labels
Actor

AU-5(1)	RESPONSE TO AUDIT PROCESSING FAILURES | *AUDIT STORAGE CAPACITY*		
	ASSESSMENT OBJECTIVE: *Determine if:*		
	AU-5(1)[1]	*the organization defines:*	
		AU-5(1)[1][a]	*personnel to be warned when allocated audit record storage volume reaches organization-defined percentage of repository maximum audit record storage capacity;*
		AU-5(1)[1][b]	*roles to be warned when allocated audit record storage volume reaches organization-defined percentage of repository maximum audit record storage capacity; and/or*
		AU-5(1)[1][c]	*locations to be warned when allocated audit record storage volume reaches organization-defined percentage of repository maximum audit record storage capacity;*
	AU-5(1)[2]	*the organization defines the time period within which the information system is to provide a warning to the organization-defined personnel, roles, and/or locations when allocated audit record storage volume reaches the organization-defined percentage of repository maximum audit record storage capacity;*	
	AU-5(1)[3]	*the organization defines the percentage of repository maximum audit record storage capacity that, if reached, requires a warning to be provided; and*	
	AU-5(1)[4]	*the information system provides a warning to the organization-defined personnel, roles, and/or locations within the organization-defined time period when allocated audit record storage volume reaches the organization-defined percentage of repository maximum audit record storage capacity.*	

Figure 2. AU-5(1) Assessment Guidelines

The NIST SP800-53A [20], companion to the 800-53, provides assessment guidelines for each security control. Figure 2 shows the guidelines for AU-5(1). Notice that it dissects the security control statement into evaluative portions, providing distinct labels for each portion. We use both the security control and its guidelines to create and instantiate a security assurance case for a specific information system using GSN.

4. Sample Application

We demonstrate security assurance case expression, evolution, and satisficing evaluation on a sample Smart Inventory Management System (SIMS). The architecture is shown in Figure 3. The components operate concurrently, each with their own MAPE-K (monitor-analyze-plan-execute-knowledge) loop which is a common control loop to perform self-adaptation. Measure collects sensor data and emits a local signal indicating if the current reading is outside a defined threshold. The sensor readings are passed to Process, a cloud service, that adjusts the sensor threshold based on the received readings. Measure and Process create audit records that are sent to Audit's message queue, which then stores them in an audit trail.

The process flow for SIMS appears in Figure 4. Process flow understanding is needed because it is possible to formally express the low-level functionality that is part of a security control and directly prove the implementation complies with it [17]. A formal proof can be part of the argument needed within a security assurance case as described in the next section.

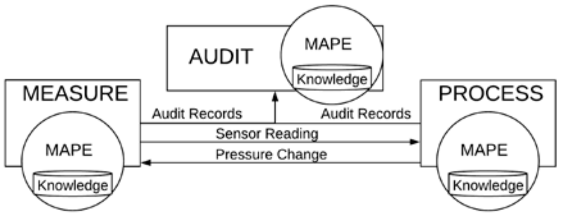


Figure 3. SIMS Architecture

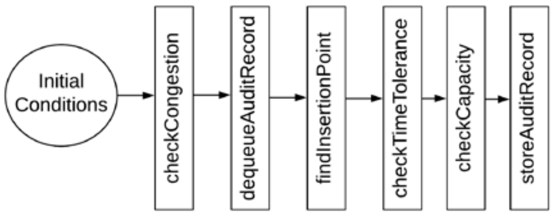


Figure 4. SIMS Audit Component Processes

The MAPE-K loop in each component monitors for anomalies in the system and activates the planner to generate an adaptation. The *checkCongestion* process in Figure 4 provides the monitor with information about the input queue. Imagine that the monitor has received a certain pattern of information from *checkCongestion* that causes it to invoke the analyze phase. Here it is determined that the input queue is filling too rapidly for Audit, but that Measure and Process are not the problems. The planner configures three potential adaptations.

A1: *Increase the capacity ratio limit*, delaying the generation of an audit trail capacity alert.

A2: *Introduce a new storage buffer and alter Audit* to offload older records in its audit trail to the new buffer.

A3: *Change Audit to overwrite old records and disable capacity alert* within the same audit trail.

We return to these adaptations after introducing the assurance cases for the security controls.

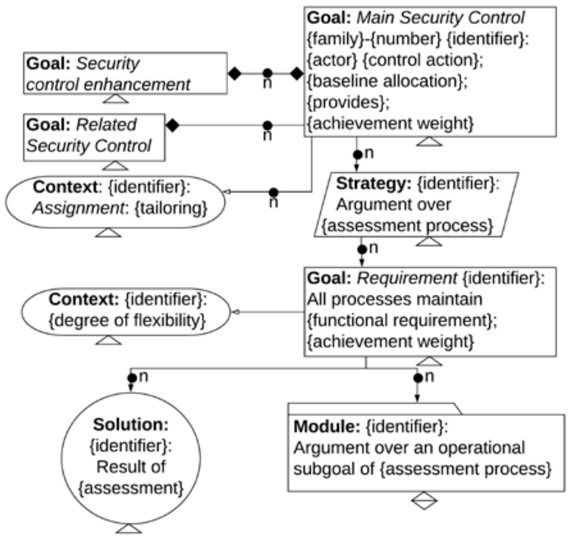

Goal: Main Security Control
{family}-{number} {identifier}:
{actor} {control action};
{baseline allocation};
{provides};
{achievement weight}
Goal: Security control enhancement
Goal: Related Security Control
Context: {identifier}: Assignment: {tailoring}
Strategy: {identifier}: Argument over {assessment process}
Goal: Requirement {identifier}: All processes maintain {functional requirement}; {achievement weight}
Context: {identifier}: {degree of flexibility}
Solution: {identifier}: Result of {assessment}
Module: {identifier}: Argument over an operational subgoal of {assessment process}
n

Figure 5. GSN Template for Security Assurance Case

5. Creating Security Assurance Cases

Given that 800-53 security controls have a similar structure as in Figure 1, we extend a basic GSN template [10] to allow for dependency and achievement weight expressions as shown in Figure 5. We instantiate the security assurance cases from the template using an approach found in [9]. The 800-53A directs the expansion of the assurance case into subgoals, context elements, and strategies for each control. Evidence can be formulated by multiple means, such as testing, model checking, and proof. We express the template in XML, based on CertWare [3] but without the use of its display facilities to allow for more coding flexibility.

In Figure 5, the root goal is the main security control with attributes that coincide with the labeling and control statement (Figures 1 and 2). The impact baseline allocation is provided. The "provides" attribute holds the *provision set* of state variables and conditions that are part of the mechanisms needed for compliance with the security control. This set flows through a SupportedBy link that is augmented with a diamond to indicate the security control source for the provision set. In Figure 5, provision sets flow to the main control from related controls and enhancements. The achievement weight, a_w, is assigned to all goals. It holds the current value calculated at the goal for assessing the satisficing level of the main goal as discussed in Section 7.

Context nodes are connected through an InContextOf link (hollow arrow). Context nodes hold the assignment tailoring as discussed in Section 3. Attached to subgoals, they may hold functional requirement weights for how flexible they are to change. The strategy connects the main goal by a SupportedBy link (filled arrow) to the assurance case argument from which the subgoals and solutions extend. Modules represent low-level operational goals for argumentation related directly to the verification and validation processes employed. The triangles mean the node is uninstantiated. The joined triangles mean the node is both undeveloped and uninstantiated.

Figure 6 instantiates the security assurance case template for AU-5(1) using 800-53A labels. The subgoal Req1 is a functional requirement represented by an invariant expressed in Linear Temporal Logic, as "it is always the case that the audit trail size is less than the capacity ratio limit associated with the record storage capacity or an alert occurs." The context nodes in the instantiation have the tailoring for *capRatioLimit* and the various alert parameters segregated in Figure 2. AU-5 holds the capacity value in its provision set for AU-5(1) that it acquires from its dependency on AU-4. AU-5(1) assigns the value of *capAlert* which it provides to AU-5. The modules M1-M6 are the operational goals related to the process flow for SIMS in Figure 4.

6. Adapting Assurance Cases

To illustrate performing and evaluating an adaptation on a security assurance case, we expand Module M5 in Figure 6 to show the argument of maintaining a satisfactory impact on the *checkCapacity* process. Figure 7 shows the expanded module for M5, which has the argument over the proof process of our system. The proof process is modeled as operational goals to maintain the invariant subgoal from Figure 6.

We assume the MAPE-K loop planner can describe the needed changes to the XML that represents the security assurance case and construct the adapted assurance cases for **A1** through **A3** as described in Section 4.

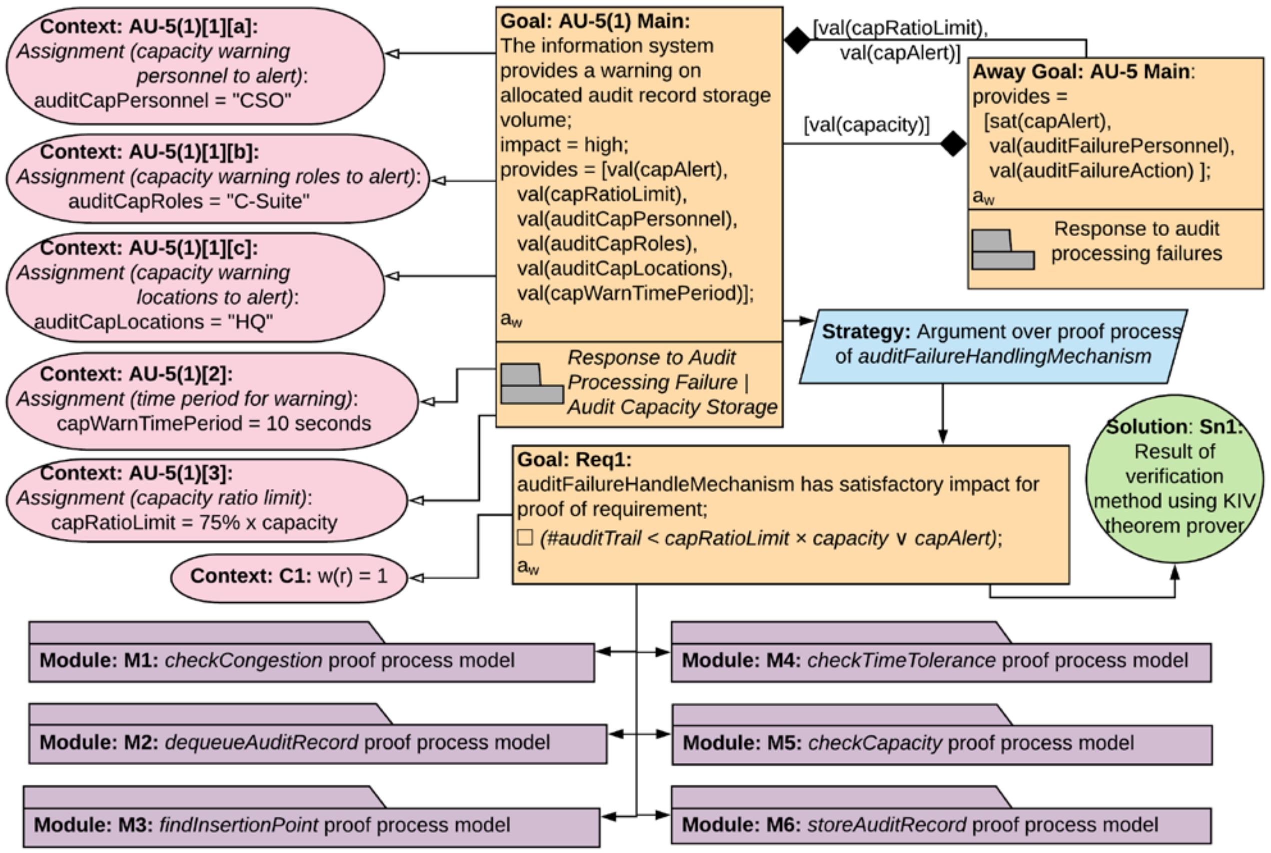

Context: AU-5(1)[1][a]: Assignment (capacity warning personnel to alert): auditCapPersonnel = "CSO"
Context: AU-5(1)[1][b]: Assignment (capacity warning roles to alert): auditCapRoles = "C-Suite"
Context: AU-5(1)[1][c]: Assignment (capacity warning locations to alert): auditCapLocations = "HQ"
Context: AU-5(1)[2]: Assignment (time period for warning): capWarnTimePeriod = 10 seconds
Context: AU-5(1)[3]: Assignment (capacity ratio limit): capRatioLimit = 75% x capacity
Context: C1: w(r) = 1
Goal: AU-5(1) Main: The information system provides a warning on allocated audit record storage volume; impact = high; provides = [val(capAlert), val(capRatioLimit), val(auditCapPersonnel), val(auditCapRoles), val(auditCapLocations), val(capWarnTimePeriod)]; aw
Response to Audit Processing Failure | Audit Capacity Storage
[val(capRatioLimit), val(capAlert)]
[val(capacity)]
Away Goal: AU-5 Main: provides = [sat(capAlert), val(auditFailurePersonnel), val(auditFailureAction)]; aw
Response to audit processing failures
Strategy: Argument over proof process of auditFailureHandlingMechanism
Goal: Req1: auditFailureHandleMechanism has satisfactory impact for proof of requirement; □ (#auditTrail < capRatioLimit × capacity ∨ capAlert); aw
Solution: Sn1: Result of verification method using KIV theorem prover
Module: M1: checkCongestion proof process model
Module: M2: dequeueAuditRecord proof process model
Module: M3: findInsertionPoint proof process model
Module: M4: checkTimeTolerance proof process model
Module: M5: checkCapacity proof process model
Module: M6: storeAuditRecord proof process model

Figure 6. Security Assurance Case for AU-5(1)

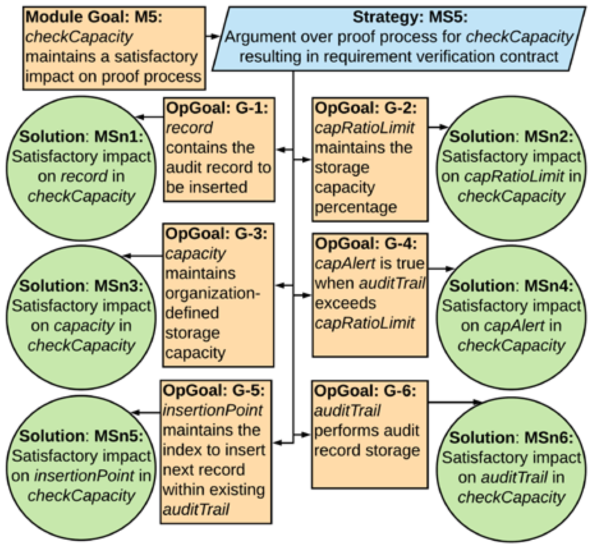
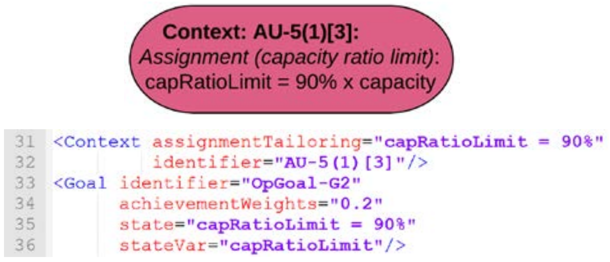

Module Goal: M5: checkCapacity maintains a satisfactory impact on proof process
Strategy: MS5: Argument over proof process for checkCapacity resulting in requirement verification contract
OpGoal: G-1: record contains the audit record to be inserted
OpGoal: G-2: capRatioLimit maintains the storage capacity percentage
OpGoal: G-3: capacity maintains organization-defined storage capacity
OpGoal: G-4: capAlert is true when auditTrail exceeds capRatioLimit
OpGoal: G-5: insertionPoint maintains the index to insert next record within existing auditTrail
OpGoal: G-6: auditTrail performs audit record storage
Solution: MSn1: Satisfactory impact on record in checkCapacity
Solution: MSn2: Satisfactory impact on capRatioLimit in checkCapacity
Solution: MSn3: Satisfactory impact on capacity in checkCapacity
Solution: MSn4: Satisfactory impact on capAlert in checkCapacity
Solution: MSn5: Satisfactory impact on insertionPoint in checkCapacity
Solution: MSn6: Satisfactory impact on auditTrail in checkCapacity

Figure 7. Expanded *checkCapacity* Module

Adaptation **A1** directly affects the assurance case for AU-5(1) by changing the *capRatioLimit* tailored value in the context node Context: AU-5(1)[3] of Figure 6. Figure 8 reflects the change to the adapted Context: AU-5(1)[3] node, where the tailored value increases from 75% to 90%. It also includes the XML for that context node where the adaptation increases *capRatioLimit* as shown on line 31. The impact to the achievement weight is shown on line 34.

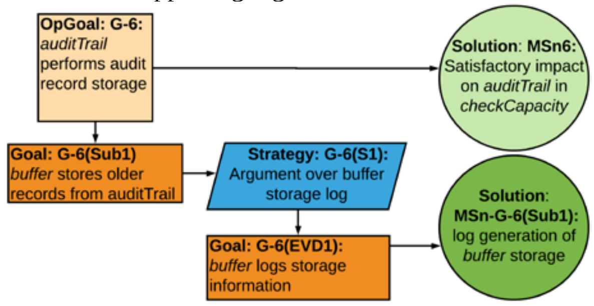

Context: AU-5(1)[3]: Assignment (capacity ratio limit): capRatioLimit = 90% x capacity

```
31 <Context assignmentTailoring="capRatioLimit = 90%"
32        identifier="AU-5(1)[3]"/>
33 <Goal identifier="OpGoal-G2"
34       achievementWeights="0.2"
35       state="capRatioLimit = 90%"
36       stateVar="capRatioLimit"/>
```

Figure 8. AU-5(1) with Adaptation A1

Adaptation **A2** introduces a new buffer into the Audit component, but AU-5(1)'s assurance case has no solution node to satisfy the new subgoal. Because there exist security controls that refer to offloading audit records to alternate storage, we assume the planner can reuse the evidence that such logging is sufficient to comply with operation goal G-6.

Figure 9 reflects the adapted operational goal G-6 from Figure 7 for adaptation **A2.** This adaptation introduces a new branch for G-6 to be satisfied with an argument using an external buffer to store older records in the audit trail through G-6(Sub1), G-6(S1), G-6(EVD1). The XML produced by the planner reflects the argument additions. Line 31 shows a reduced achievement weight to 0.5, reflecting the potential for a negative impact on the goal. The goal for the new supporting argument is added at line 34.


OpGoal: G-6:
auditTrail performs audit record storage
Solution: MSn6: Satisfactory impact on *auditTrail* in *checkCapacity*
Goal: G-6(Sub1) *buffer* stores older records from auditTrail
Strategy: G-6(S1): Argument over buffer storage log
Solution: MSn-G-6(Sub1): log generation of *buffer* storage
Goal: G-6(EVD1): *buffer* logs storage information

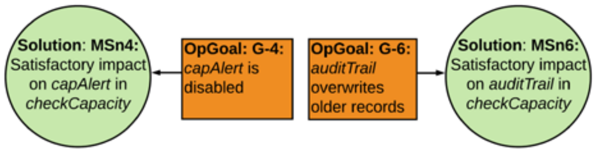
```
12 <Goal identifier="G-AU-5(1)"
13         achievementWeights="0.7777777777777778"
14         family="AU"  number="5"
15         actor="IS" baselineAllocation="high"
16         controlAction="a warning on allocated
17                        audit record storage volume"
18         provides="val(capRatioLimit);
19                   val(auditCapPersonnel);
20                   val(auditCapRoles);
21                   val(auditCapLocation);
22                   val(capWarntimePeriod)"
23         requires="val(capRatioLimit);
24                   val(capacity)"/>
25 <Goal identifier="M5"
26         achievementWeights="0.8333333333333334"
27         assessmentProcess="checkCapacity maintains
28                            satisfactory impact on
29                            proof process"/>
30 <Goal identifier="OpGoal-G6"
31         achievementWeights="0.5"
32         state="auditTrail offload older record"
33         stateVar="auditTrail"/>
34 <Goal identifier="OpGoal-G6(sub1)"
35         achievementWeights="0.5"
36         state="store older record"
37         stateVar="buffer"/>
```

Figure 9. AU-5(1) with Adaptation A2

Figure 10 shows the affected operational goals G-4 and G-6 from Figure 7 due to adaptation **A3**. The adaptation affects G-6 and G-4 by substituting their functions with overwriting older records and disabling the capacity alert, respectively, to satisfy module M5's goal. The XML lines 31and 35 indicate the reduced achievement weights to 0.2 that impact the goal specified in line 12.

7. Goal Satisficing Level Determination using Achievement Weights

Maintaining the security control in the self-adaptive system is a non-functional requirement. Using concepts discussed in Section 2, we represent each main security control as a softgoal and use the subgoals and operational goals from its security assurance case to create direct edges that form a Softgoal Interdependency Graph (SIG) [18]. The SIG results in a tree with only AND relationships. We adapt the Soft Goal using Weight (SGW) approach [12] to determine the satisficing level of the assurance case. A modified vulnerability metric calculation [24] provides the achievement weight of each softgoal. Satisficing calculations can indicate the impact of an adaptation on the security assurance case, including propagation of required state values from other security controls. The remainder of the section defines the formulas and their adaptations from the original approaches [12, 24]. We show how the achievement weights and satisficing levels are calculated for adaptations **A1-A3** and the level of satisficing that results from each.


Solution: MSn4: Satisfactory impact on *capAlert* in *checkCapacity*
OpGoal: G-4: *capAlert* is disabled
OpGoal: G-6: *auditTrail* overwrites older records
Solution: MSn6: Satisfactory impact on *auditTrail* in *checkCapacity*

```
12 <Goal identifier="G-AU-5(1)"
13         achievementWeights="0.7111111111111111"
14         family="AU" number="5"
15         actor="IS" baselineAllocation="high"
16         controlAction="a warning on allocated
17                        audit record storage volume"
18         provides="val(capRatioLimit);
19                   val(auditCapPersonnel);
20                   val(auditCapRoles);
21                   val(auditCapLocation);
22                   val(capWarntimePeriod)"
23         requires="val(capRatioLimit);
24                   val(capacity)"/>
25 <Goal identifier="M5"
26         achievementWeights="0.7333333333333334"
27         assessmentProcess="checkCapacity maintains
28                            a satisfactory impact on
29                            proof process"/>
30 <Goal identifier="OpGoal-G4"
31         achievementWeights="0.2"
32         state="capAlert = disable"
33         stateVar="capAlert"/>
34 <Goal identifier="OpGoal-G6"
35         achievementWeights="0.2"
36         state="auditTrail overwrite older record"
37         stateVar="auditTrail"/>
```

Figure 10. AU-5(1) with Adaptation A3

Using the SGW approach [12], we define a softgoal interdependency graph, SIG_A, for the security assurance case, A, as a tree of goals with the main security goal, m_A, as the root. $SIG_A = (G_A, D_A)$ where

- $G_A = \{m_A\} \cup O_A$
- O_A = set of subgoals and operational goals for A that support the main security control, m_A (root)
- For all goals $g \in G_A$, $a_w(g)$ is the achievement weight calculated for that goal.
- D_A = the set of edges (p, c), representing dependencies among the parent (p) and child (c) goals in SIG_A.

The related security controls introduce inter-dependencies that form a network of security controls. As assurance cases, they only have knowledge of the controls on which they depend. However, from the

MAPE-K loop perspective, the inter-dependencies can be traversed as an adaptation is evaluated. A partial dependency graph appears in Figure 11. The links specify the provision sets passed from source (diamond) to target control. This expression facilitates the propagation impact evaluation of an adaptation.

The security control network (SCN) = (M, D_M), where

- M = $\{\bigcup_{\mathrm{SIG}} m\}$, the set of all SIG root goals
- D_M = set of weighted, directed edges with provision sets representing dependencies among security controls (Figure 11).

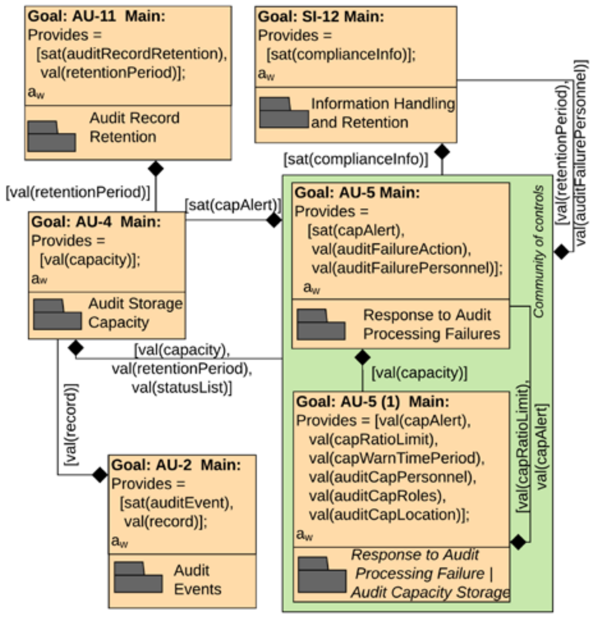

Goal: AU-11 Main:
Provides = [sat(auditRecordRetention), val(retentionPeriod)];
aw
Audit Record Retention
Goal: SI-12 Main:
Provides = [sat(complianceInfo)];
aw
Information Handling and Retention
[sat(complianceInfo)]
[val(retentionPeriod)]
[sat(capAlert)]
[val(retentionPeriod), val(auditFailurePersonnel)]
Goal: AU-4 Main:
Provides = [val(capacity)];
aw
Audit Storage Capacity
Goal: AU-5 Main:
Provides = [sat(capAlert), val(auditFailureAction), val(auditFailurePersonnel)];
aw
Response to Audit Processing Failures
Community of controls
[val(capacity), val(retentionPeriod), val(statusList)]
[val(capacity)]
[val(record)]
Goal: AU-2 Main:
Provides = [sat(auditEvent), val(record)];
aw
Audit Events
Goal: AU-5 (1) Main:
Provides = [val(capAlert), val(capRatioLimit), val(capWarnTimePeriod), val(auditCapPersonnel), val(auditCapRoles), val(auditCapLocation)];
aw
Response to Audit Processing Failure | Audit Capacity Storage
[val(capRatioLimit), val(capAlert)]

Figure 11. Sample Security Control Network

The community structure advocated by the approach in [24] provides for a higher degree of influence across the edges. In our representation, a security control and its enhancements form a natural community, as represented by the green box surrounding AU-5 and AU-5(1) in Figure 11. To calculate $a_w(g)$ for $g \in$ G_A, we measure the vulnerabilities of the community structure in the SCN. The achievement weight is inversely related to community vulnerability as described in [24].

Achievement weight is then defined for a SIG_A as

$$\begin{aligned} a_w(g) &= I(g), \text{ for leaf nodes, } g \in O_A \\ &= average\big(a_w(c)\big), \text{ for all } c \in children(g) \\ &\qquad \text{for non-leaf nodes, } g \in G_A \end{aligned}$$

where $I(g)$ is the impact factor defined on the state variables supporting the operational goals at the SIG leaves. Currently, $I(g)$ must be determined by the certifiers prior to deployment given potential changes to state variables and the organization's risk policy.

Table 1 provides sample values for $I(g)$ related to the state variables affects by adaptations **A1-A3**. A lower value has more negative impact on achievement weights. In a community, the control enhancements (e.g. AU-5(1)) propagate their achievement weights to their community parent (e.g. AU-5) as one of its edges.

Table 1. Sample Impact Table

$I(g)$	*capRatio-Limit*	*capacity*	*auditTrail*	*insertion-Point*
1	= 75 %	= 100	Store record	≤ *#records*
0.9		> 100		
0.5	< 75%	< 100	Offload older record	> *#records*
0.2	> 75%		Overwrite older record	
0	≤ 0% or ≥ 100%	≤ 0	Drop record	< 0

Determining the satisficing level of a main control softgoal, such as AU-5, relies on the SCN. A partial SCN is shown in Figure 11. The satisficing level, $SL(m)$, of main goal m is the average of achievement weights that include $a_w(m)$ and the $neighbors(m)$ as defined by the direction that the provision sets are passed. For example, $neighbors$(AU-4) = {AU-2, AU-5, AU-6, AU-7, AU-11, SI-4} from Figure 1, with a subset shown in Figure 11. Thus,

$$SL(m) = average\big(a_w(m) + \textstyle\sum_{g \,\in\, neighbors(m)} a_w(g)\big)$$

A control enhancement, e, that has a neighbor outside of its community can potentially have $SL(e) \neq a_w(e)$. In this case, $SL(e)$ has priority. When security controls are mutually related with the same provision, the algorithm cannot double count the impact. To resolve this issue, our satisficing algorithm preserves the last calculated achievement weight, $a_{prev}(g)$, and uses that achievement weight as the neighbor's achievement weight to stabilize the network-based calculation. We assume that when deployed, the SIMS security controls have an achievement weight of 1. We show how adaptations **A1-A3** directly lower certain achievement weights and propagate the impact through the SCN.

8. Adaptation Results

Table 2 shows the achievement weight changes for AU-5's partial community after applying adaptations **A1-A3** to the security assurance case for AU-5(1). Though we focused on Module M5, other modules are also affected by the adaptations and are reflected in Table 2.

Table 2. $a_w(g)$ in AU-5 Community

Goal	Base	A1	A2	A3
Opp-G1	1	0.2	1	1
Opp-G2	1	1	1	1
Opp-G3	1	1	1	1
Opp-G4	1	1	1	0.2
Opp-G5	1	1	0.5	1
Opp-G6	1	1	0.5	0.2
M1	1	1	1	1
M2	1	1	0.5	0.2
M3	1	1	0.5	0.6
M4	1	1	1	1
M5	1	0.867	0.833	0.733
M6	1	0.867	0.833	0.733
G1	1	0.956	0.778	0.711
AU-5(1)	1	0.956	0.778	0.711
AU-5	1	0.956	0.778	0.711

Table 3 shows the satisficing level computed for each main security control at the base (deployed) level and after applying adaptations **A1-A3**. Note that SL(AU-5(1)) = a_w(AU-5(1)) because the **A1-A3 a**re internal to that security control. AU-5 is affected by **A1-A3** because of its relationship with AU-5(1). The effects of **A1** and **A2** only propagate to AU-5 since the adapted provisions remain in the community. Adaptation **A3** impacts AU-5 and AU-4 because *capAlert* is in the propagated provision set (Figure 11).

Table 3. Satisficing Levels

	Base	A1	A2	A3
AU-2	1	1	1	1
AU-4	1	1	1	0.928
AU-5	1	0.985	0.926	0.903
AU-5(1)	1	0.956	0.778	0.711
AU-11	1	1	1	1
SI-12	1	1	1	1

9. Adaptation Evaluation

To evaluate the alignment of the adaptive system behavior with the satisficing level determination in Section 8, we deploy **A1-A3** in the SIMS application. We embed checkpoints as probes in the *checkCapacity* module (M5 in Figures 6 and 7) and log the effects on the audit trail. Figure 12 shows how the checkpoints are placed to determine if (*i*) a *record* is generated (CK1), (*ii*) the *capRatioLimit* is maintained (CK2), (*iii*) the audit trail *capacity* is maintained with capability to store a record within the *auditTrail* (CK3), (*iv*) the alert is properly performed by *capAlert* (CK4), (*v*) the proper *insertionPoint* can be found to store the next record while maintaining the existing *auditTrail* contents (CK5), and (*vi*) the record is stored in *auditTrail* (CK6).

We ran tests with sufficient audit trail capacity and insufficient audit trail capacity. With sufficient capacity, adaptation **A1** performs better than **A2** and **A3**. Allowing more records to flow into the audit trail is a local change that impacts only a single state variable and does not propagate outside the community. Thus, **A1** is not heavily relied on by the assurance case argument or proof for all audit functionality. **A2** and **A3** impact several operational goals that are needed for the overall argument or proof. Table 4 shows the results with insufficient capacity in which the audit trail can hold only 50 records. Column 1 represents the base deployment (B), followed by the adaptations when the number of records needed is 75 and 100. **A1** does poorly with insufficient records. **A2** performs the best but requires addition buffer storage. **A3** performs worse than **A1** overall. **A3** fails at CK4 by disabling *capAlert* and fails at CK5 when overwrite functionality violates the requirement that the insertion point maintains the records in the audit trail.

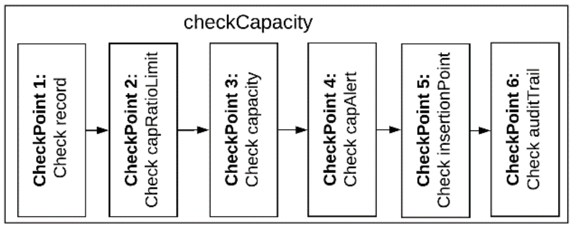

checkCapacity
CheckPoint 1: Check record
CheckPoint 2: Check capRatioLimit
CheckPoint 3: Check capacity
CheckPoint 4: Check capAlert
CheckPoint 5: Check insertionPoint
CheckPoint 6: Check auditTrail

Figure 12. *checkCapacity* Checkpoints

Table 4. Performance Evaluation Results

	#Rec	CK1	CK2	CK3	CK4	CK5	CK6
B	75	75	37	50	75	50	50
B	100	100	37	50	100	50	50
A1	75	75	43	50	75	50	50
A1	100	100	43	50	100	50	50
A2	75	75	74	50	75	75	75
A2	100	100	99	50	100	100	100
A3	75	75	74	50	37	37	75
A3	100	100	99	50	37	37	100

10. Limitations and Future Work

In this paper, we extend a security assurance case template to specify goal achievement weights and interdependencies among a network of related security controls. The specification introduces the calculation of a satisficing level of a security control for a potential self-adaptation based on its internal changes and from propagated satisficing levels in the network. We implement the security assurance cases using XML to perform the adaptations and measurements at runtime, as demonstrated using a sample application with three adaptations and embedded checkpoints. We discuss the alignment of the adaptation failure rates with the calculated satisficing levels. Using system domain knowledge, experts can introduce satisficing level thresholds to identify acceptable adaptations.

Scalability is a potential limitation to the approach given the size of the security control network of related controls for a large-scale system. The XML representation can streamline the automated assessment process when an adaptation is considered. Though codifying the security assurance cases in XML is potentially burdensome during design, once codified, achievement weight and satisficing level determination could be optimized. Evaluating scalability will be part of future work, which will also examine patterns of applications and adaptations to determine the influence the presumed dependencies actually have on related controls.

Acknowledgement. This material is based on research sponsored by Air Force Research Laboratory under agreement number FA8750-16-1-0248. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory or the U.S. Government.

11. References

[1] M. Anisetti, et al., “A certification framework for cloud-based services,” SAC, 2016.

[2] C. A. Ardagna, et al., "A Certification Technique for Cloud Security Adaptation," IEEE Int’l. Conf. on Services Computing, pp. 324-331, 2016.

[3] CertWare, https://nasa.github.io/CertWare/

[4] S. W. Cheng, D. Garlan, and B. R. Schmerl, “Evaluating the Effectiveness of the Rainbow Self-Adaptive System”, IEEE SEAMS, pp. 132-141, 2009.

[5] L. Chung, and B. A. Nixon, “Dealing with Non-Functional Requirements: Three Experimental Studies of a Process-Oriented Approach”, 17th Int’l Conf. on Software Engineering, pp. 24-28, 1995.

[6] M. Felici, “Evolutionary Safety Analysis: Motivations from the Air Traffic Management Domain”, Int’l Conf. on Computer Safety, Reliability, and Security, Springer-Verlag, pp. 208-221, 2005.

[7] M. Felici, “Modeling Safety Case Evolution – Examples from the Air Traffic Management Domain”, Int’l Wksp on Rapid Integration of Soft. Eng. Techniques, Springer-Verlag, pp. 81-96, 2005.

[8] J. Goodenough, H.F. Lipson, and C.B. Weinstock, “Arguing Security – Creating Security Assurance Cases”, US Computer Emergency Readiness Team – Build Security In, 2007.

[9] R. Hawkins, et al., “Weaving an Assurance Case from Design: A Model-Based Approach,” IEEE 16th Int’l. Symp. on High Assurance Systems Engineering, 2015.

[10] S. Jahan, A. Marshall, and R. Gamble, “Self-Adaptation Strategies to Maintain Security Assurance Cases”, 12th IEEE International Conference on Self-Adaptive and Self-Organizing Systems, 2018

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

[12] N. Kobayashi, et al., “Quantitative Non-Functional Requirements Evaluation Using Softgoal Weight”, J. of Internet Services and Information Security, Institute of Engineering – Polytechnic of Porto, 6:1(37-46), 2016.

[13] S. Kokaly, et al., “A model management approach for assurance case reuse due to system evolution”, ACM/IEEE 19th Int’l Conf. on Model Driven Engineering Languages and Systems, 2016.

[14] C. Le Goues, et al., “GenProg: A Generic Method for Automatic Software Repair”, IEEE Trans. on Software Engineering, 38:1(54-72), 2011.

[15] C. Lin, et al., “Measure Confidence of Assurance Cases in Safety-Critical Domains”, Proc. of the 40th Int’l Conf. on Soft. Eng., pp. 13-16, 2018.

[16] H.F. Lipson and C. B. Weinstock, “Evidence of Assurance: Laying the Foundation for a Credible Security Case”, available at https://www.us-cert.gov/bsi/articles/knowledge/assurance-cases/evidence-assurance-laying-foundation-credible-security-case, 2008.

[17] A. Marshall, S. Jahan, and R. Gamble, “Toward Evaluating the Impact of Self-adaptation on Security Control Certification,” 13th Int’l Conf. on Soft. Eng. for Adaptive and Self-Managing Systems, 2018.

[18] J. Mylopoulos, L. Chung, and B. Nixon, “Representing and Using Nonfunctional Requirements: A Process-Oriented Approach”, IEEE Trans. on Soft. Eng., 18:6(483-497), 1992.

[19] NIST, “Security and Privacy Controls for Federal Information Systems”, NIST Special Publication 800-53, Revision 4, 2013.

[20] NIST, Assessing Security and Privacy Controls in Federal Information Systems and Organizations”, NIST Special Publication 800-53A, Revision 4, 2014.

[21] R. Ross, et al. “Protecting Controlled Unclassified Information in Nonfederal Information Systems and Organization,” NIST SP800-171, 2015.

[22] J. Rushby, “The Interpretation and Evaluation of Assurance Cases”, Technical Report SRI-CSL-15-01, SRI International, 2015.

[23] C. Walter, et al., “Toward Predicting Secure Environments for Wearable Devices”, Proc. of the 50th Hawaii Int’l Conf. on System Sciences, 2017.

[24] D. Wei, X. Zhang, and S. Mahadevan, “Measuring the vulnerability of community structure in complex networks”, Reliability Engineering and System Safety, Vol. 174, pp. 41-52, 2018.