12-30-2017

Semantic hierarchies for extracting, modeling, and connecting compliance requirements in information security control standards

Matthew L. Hale
University of Nebraska at Omaha, mlhale@unomaha.edu

Rose F. Gamble
University of Tulsa

Requirements Engineering
https://doi.org/10.1007/s00766-017-0287-5

ORIGINAL ARTICLE

Semantic hierarchies for extracting, modeling, and connecting compliance requirements in information security control standards

Matthew L. Hale[1] · **Rose F. Gamble**[2]

Received: 4 May 2017 / Accepted: 18 December 2017

Abstract
Companies and government organizations are increasingly compelled, if not required by law, to ensure that their information systems will comply with various federal and industry regulatory standards, such as the NIST Special Publication on Security Controls for Federal Information Systems (NIST SP-800-53), or the Common Criteria (ISO 15408-2). Such organizations operate business or mission critical systems where a lack of or lapse in security protections translates to serious confidentiality, integrity, and availability risks that, if exploited, could result in information disclosure, loss of money, or, at worst, loss of life. To mitigate these risks and ensure that their information systems meet regulatory standards, organizations must be able to (a) contextualize regulatory documents in a way that extracts the relevant technical implications for their systems, (b) formally represent their systems and demonstrate that they meet the extracted requirements following an accreditation process, and (c) ensure that all third-party systems, which may exist outside of the information system enclave as web or cloud services also implement appropriate security measures consistent with organizational expectations. This paper introduces a step-wise process, based on *semantic hierarchies*, that systematically extracts relevant security requirements from control standards to build a certification baseline for organizations to use in conjunction with formal methods and service agreements for accreditation. The approach is demonstrated following a case study of all audit-related controls in the SP-800-53, ISO 15408-2, and related documents. Accuracy, applicability, consistency, and efficacy of the approach were evaluated using controlled qualitative and quantitative methods in two separate studies.

Keywords Security policy · Security requirements · Requirement extraction · Security control standards · Regulatory compliance · Certification · Accreditation · Semantic hierarchy

1 Introduction

Laws, regulations, and corporate policies increasingly require companies and government organizations to demonstrate that mission or business critical information systems and IT infrastructures satisfy a set of *security policies* governing user behavior, system behavior, and emergency fail-safes. Given the critical nature of these systems, a lack or lapse of security protections translates to serious privacy and confidentiality risks that, if exploited, could result in information disclosure, loss of money, or, at worst, loss of life. To mitigate these risks and ensure that their information systems meet regulatory standards, organizations must be able to (a) contextualize regulatory documents in a way that extracts and expresses all of the relevant technical security implications for their systems, (b) formally certify that their in-house systems meet the extracted requirements, and (c) ensure that all integrated third-party systems, which may exist outside of the protected information system enclave as web services in a cloud, also implement appropriate security measures consistent with the regulatory documents. Certification practices require expertise in the information system domain, as well as understanding the corporate culture and organization expectations for security certification. Implemented as *information system security controls*, the technical portion of security policies typically include, at a minimum, provisions governing access control, audit, data protection,

✉ Matthew L. Hale
mlhale@unomaha.edu

Rose F. Gamble
gamble@utulsa.edu

[1] Nebraska University Center for Information Assurance, University of Nebraska at Omaha, Omaha, NE, USA

[2] Tandy School of Computer Science, University of Tulsa, Tulsa, OK, USA

Published online: 30 December 2017

contingency planning, and non-repudiation [1–5]. Certification experts seek to re-use techniques, as evidenced by the overlay concepts now highlighted in the NIST 800-53r4 [2], which designate specific groups of controls, beyond the baselines, that can be used for sector or industry applications such as health care, avionics, or critical infrastructure systems.

Occurring in tandem with system development and integration, the verification step of software system certification requires that system developers and security analysts examine security control documentation to identify all applicable *security requirements* and then extensively test the system to determine whether or not it satisfies the requirements [6–9]. Currently, examining and contextualizing regulatory documents means that developers and security analysts must read through, decompose, and interpret long, complex, natural language textual security control documents, such as the NIST SP-800-53 [2] (for federal information systems), the Health Information Portability and Accountability Act (HIPAA) (for medical information systems) [10], or the Common Criteria (ISO-15408 for industry information systems) [3] without formal underpinnings [11]. Based on the interpreted information, expansive test cases and security checklists must then be generated to test all system component features [1, 12] and certify that they meet the regulatory requirements.

System components may be developed in-house by the organization, created specifically for the organization by an external organization, deployed as third-party commercial off the shelf (COTS) products, or provided as cloud-based web services. Any and all information system updates, patches, or API (i.e., application programming interface) changes that affect the system require organizations to re-examine security control documentation to determine if any security requirements were impacted. With externally provided components or third-party cloud web services, re-certification is particularly difficult, since it is more difficult to know how the changes impact the security posture of the system. As time goes on, and security controls are repeatedly reexamined, test-cases tend to become myopically focused on the new features and not the overall picture of the system [6]. Verification and certification becomes more and more piecemeal, incomplete, and inconsistent with previous system iterations.

Achieving a coherent, consistent perspective of technical security control compliance that is resilient to system changes requires coupling a requirement extraction process that uniformly expresses and organizes security requirements with formal techniques for representing information systems so that security concerns can be partitioned without fear of myopia—since all of the direct and indirect relationships are understood between component features and compliance requirements. Embedding both resilience and formalization into the Risk Management Framework [13] can enable clearer certification processes during system extensions, continuous monitoring that may identify new risks, and new application development. Capturing traceability information about how certification was performed can lead to less ambiguity and, instead, point more directly to security controls affected by extension and new emerging risks. For new development, certifiers rely on prior experience (themselves or initial consulting services) and earlier organization documents for certification of other information systems. Common controls defined for the organization or overlays of controls defined for the information system domain that have explicit formal specification and interrelationships with other controls provide this immediate traceability and direct application as new development proceeds. This paper introduces a step-wise pattern-based requirement extraction and formalization approach that produces a *compliance model* that formally represents the security compliance requirements embedded in an arbitrary group of security controls selected by organizational security policies as a set of *compliance predicates* divided into *security control semantic hierarchies*.

Forming a hierarchy begins by examining an organization's *security profile*. The profile dictates which technical security controls the organization's information systems must comply with. Each control states one or more functional *security requirements*, i.e., the organization, system, or component in the system must do something or have a certain property. Understanding how security requirements mandate compliance requirements requires the ability to identify key information in the security requirements that must be verified. To do this, our model includes a set of *security governance patterns* capable of extracting and organizing control information. The governance patterns are predicated on the understanding that security controls have similar semantics despite having disparate representations, formats, and groupings across documents.

After the security controls are extracted using the governance patterns, a *formal schema* is used to map extracted control content into formal constructs that uniformly represents information based on the type of pattern used. This step produces *compliance requirements* that dictate constraints that information systems must follow. Examining such requirements in isolation is not sufficient. Thus, in addition to the governance patterns and formal schema, we define a set of *semantic relations* capable of relating similar compliance requirements together. Semantic relations allow control compliance constraints to be properly placed in relation to compliance requirements from other security controls. Figure 1 demonstrates how this trifecta, i.e., patterns, schema, and relations, allows regulatory requirements to be extracted, formalized, and grouped into *semantic control hierarchies* that unambiguously defines a set of *compliance*

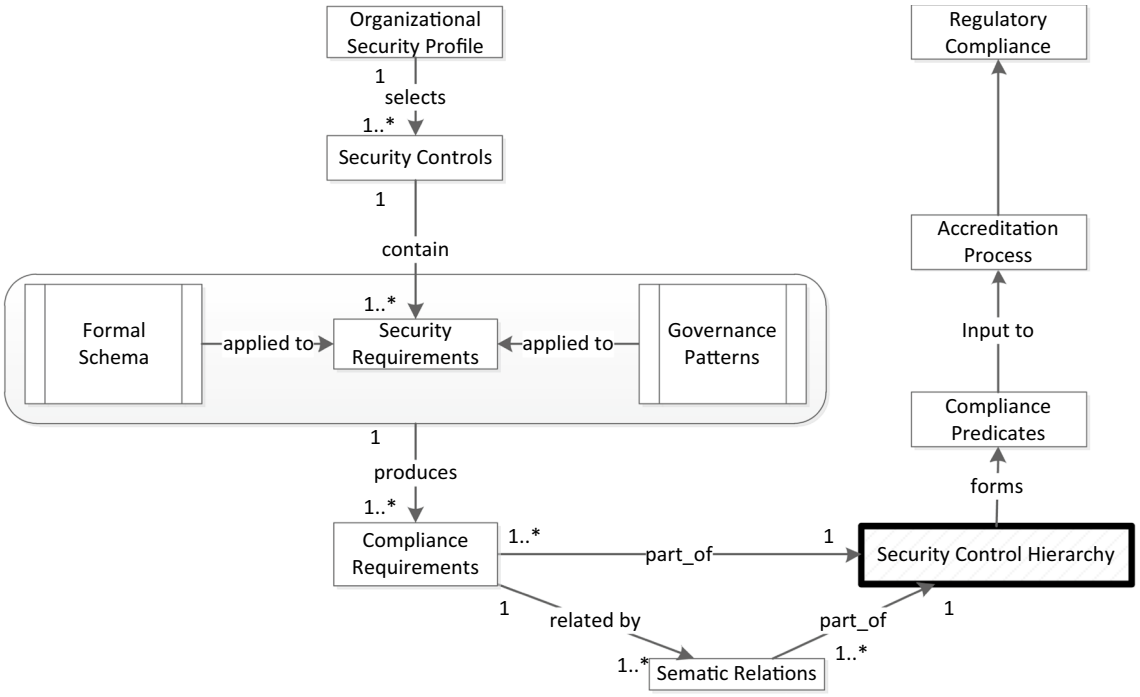

Fig. 1 Formation of the security compliance model for accreditation

predicates that completely represent organizational compliance burdens.

At the root of each control hierarchy is a single compliance predicate that expresses the functional security constraints of a collection of security controls that semantically appear below it. Children of the root may add structure, specificity, and parameters to the constraints in the predicate. Selecting a compliance predicate to be part of an information system's security profile means that a verifier can traverse the associated hierarchy, through the semantic links, to determine what branches should be verified to demonstrate compliance. Each hierarchy may encompass controls from other different control families, as well as other governing documents. Thus, the verification step for one document can show compliance with constraints in other documents. In addition, as common control groups are established for an organization's information systems and as overlays are designated for certain domains, the semantic links can point to controls that have strong interrelationships with controls in the common or overlay groups but may have been inadvertently omitted.

The compliance modeling process reduces the burden security certification by formulating compliance predicates as logical expressions that represent underlying security policy requirements in ways that are amenable to verification, can be contextualized for a particular organization, and can easily be used by other organizations following the same regulatory standard. The process also makes explicit all of the connections between disparate compliance predicates, so that there is no question as to what must be recertified when a system component is patched or changed by the organization or a third-party vendor. This rest of this paper uses a running audit case study based on the entire set of technical security controls relating to audit, taken from the SP-800-53 [2], the Common Criteria-Part 2 [3], and other regulatory standards including the DoD 8500.2 [4] and its companion Application Security and Development STIG [5]. This case study illustrates the derivation and application of the compliance model.

The rest of the paper is organized as follows: Sect. 2 overviews relevant regulatory documents, the accreditation tooling ecosystem around them, security requirement extraction techniques, and formal methods that can be used in conjunction with the compliance model for system certification. Section 3 details the three types of governance patterns and formalization templates used for requirement extraction and formalization. Section 4 defines the five semantic relations necessary to relate formalized compliance predicates. Section 5 brings the audit case study together, defining every audit-centric compliance predicate in the family of regulatory standards previously identified. Section 6 describes the methodology and results of two complementary studies with academic and industry compliance experts conducted to evaluate the approach. Finally, Sect. 7 concludes the work by discussing the

implications and applicability of this approach within the certification and accreditation tooling landscape.

2 Background

Underlying holistic security certification is the notion of managing compliance over the *life cycle* of organizational information systems by implementing and managing certain security measures to mitigate *vulnerabilities* and, thus, limit *risk*. Based on the Risk Management Lifecycle and Security Allocation Control processes in [2, 13], an *internal certification lifecycle* [14] is depicted in Fig. 2 that describes the security lifecycle of internally managed organizational systems, i.e., traditional systems without cloud web service components. The lifecycle begins when the organization defines a *security policy*. A *security policy* [2, 9] is a high-level document that addresses organizational security goals as a series of abstract, but ideally, unambiguous goal statements, e.g., user data will remain confidential or systems will remain operational 99% of the time. These goal statements then direct the selection of *security controls*, as shown in Fig. 2, from the set of security controls provided by a chosen regulatory standard (discussed in detail in Sect. 2.1). Once selected, the controls govern the design of the system, as the organization must implement the controls on top of their system functionality.

After (and during) the system design process, vulnerabilities must be assessed and mitigated in order for the security controls to be satisfied and, thus, for the organization to be in compliance with the regulatory standard. Resources such as the Common Weakness Enumeration (CWE) [15] and the Common Vulnerability Enumeration (CVE) [16] may be used to direct the assessment and mitigation of identified implementation issues. The end result of successful certification is the formation of a *secure system enclave* [2] around the data and functionality that the system contains in a way that complies with the regulatory mandates and protects critical information or infrastructure.

Organizations utilizing third-party cloud services as part of their information system processing must follow a different certification lifecycle [14], as introduced in Fig. 3. This lifecycle may be applicable for any outsourced service, but our focus is on the expected use of cloud services over which the organization has some control. In this process, the organizations are not implementing the system design. Instead they are seeking services that perform certain functions, but that implement organizationally selected security controls. In this way, an organization's choice of security controls direct the formation of a service request. The request could be for one or many web services. Once formed, the request must be assessed to determine what types of vulnerabilities might apply to it, e.g., vulnerabilities described in [15, 16]. For instance, a service request for *web composition* of cloud storage services with a point of sale system might have the potential for data disclosure at the interchange points. By identifying these vulnerabilities, the organization can construct a *risk-weighted* list of *service terms* [14] that any prospective service providers must be able to meet. A matchmaking algorithm may then be applied to examine the risk-weighted service request against actual published service provider risk terms and select the service provider with the closest match, i.e., lowest risk of non-compliance.

For many organizations the reality is often somewhere in between these two lifecycle views. Such organizations may utilize cloud-based web services combined with traditional in-house systems that form a *hybrid system* [17]. In all cases, it is important that security controls are well defined and unambiguous so that certification is consistent and repeatable [2] regardless of whether the operational environment is in the cloud or an in-house information technology (IT) asset. The next sections discuss governing regulatory documents (Sect. 2.1), the state of the art in requirements extraction and modeling (Sect. 2.2), and formal modeling

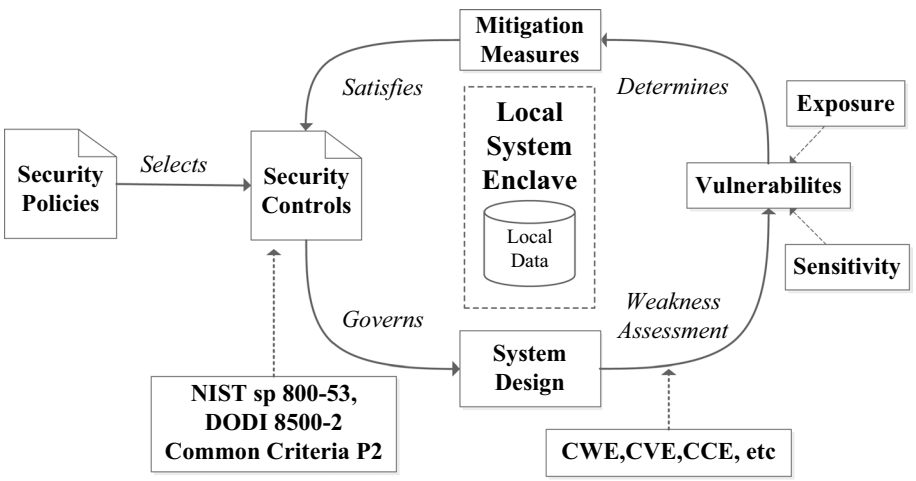

Fig. 2 Internal system certification lifecycle

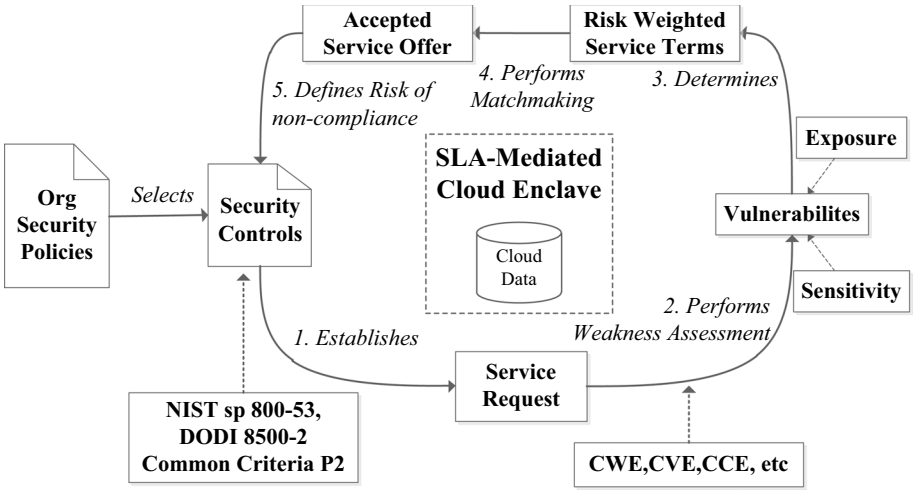

Fig. 3 External cloud-based system certification lifecycle

approaches for representing traditional and cloud-based systems (Sect. 2.3).

2.1 Federal and industry regulatory documents

Organizations with critical systems, such as government agencies, hospitals, corporations, or military branches, typically follow one or more federal or industry regulatory standards to ensure their systems meet confidentiality, integrity, and availability constraints. These regulatory standards include the NIST Recommended Security Controls for Federal Information Systems and Organizations (NIST SP-800-53) [2], Department of Defense Instruction 8500.2: Information Assurance (IA) Implementation (DoDI 8500.2) [4], and its companion document the Defense Information Systems Agency's Application Security and Development Security Technical Implementation Guide (DISA-AppStig) [5] (for federal information systems), the Health Information Portability and Accountability Act (HIPAA) (for medical information systems) [10], and the Common Criteria (ISO-15408) [3] and Cloud Computing Matrix [18] (for industry information systems). All of these documents are directly addressed in our approach with the exception of HIPAA, which has been the focus of a number of other research initiatives including widely accepted work by Breaux and Anton [19], discussed in Sect. 2.2.

The gold standard for federal information systems is the NIST SP800-53 [2]. The SP800-53 structures its security controls using a well-defined three level system. At the top is the *control family*. Each family describes either a set of controls that are *technical* (impacts system design), *management* (governs organizational planning, risk assessment and authorization), or *operational* (relates to personnel training, physical configuration and security, incident response, etc.). There are a total of 18 high-level families that include provisions for audit, access control, and contingency planning. Below the high-level family are individually numbered and named *security controls* that describe particular security requirements. Finally, the bottom level is comprised of *control enhancements* that decompose the control into particular facets of interest and generally expand on the security control content.

The DoDI 8500.2 [4] is an older federal standard, on which the Department of Defense Information Assurance Certification and Accreditation Process (DIACAP) [1] is based. The DoDI 8500.2 is much less structured than the SP800-53 and has much less concrete control statements. Each information assurance control in the DoDI 8500.2 has a unique control number and is grouped into one of several possible *subject areas*. Each control has three versions, one for each of the *mission assurance categories* which describe the level of mission criticality applicable to the control. STIGs, or *security technical implementation guidelines*, specifically the 2011 Application Security and Development STIG [5], extend the DoDI 8500.2, reducing the abstraction of control statements by filling in details such as stating the type of encryption required, or specifying the structure or type of events that must be audited. A hodgepodge of other documents, including the DoDI 5200.2, 5200.40, 8500.1, OMB-Circular A-1-130 [20–23], relate to the DoDI 8500.2, but are either to high level [22, 23], non-technical [20], or outdated [21] and are thus not applicable to our work. For instance, the DoDI 5200.2 [20] is a non-technical document describing the DoD recommendations for personnel security programs, and OMB A-130 [23] and DoDI 8500.1 [22] describe the roles of executive departments and agency heads and high-level policy requirements, respectively.

The Common Criteria for Information Technology Security Evaluation, hitherto labeled *Common Criteria* or simply *CC*, contains three parts [24]. Part 1 describes the general model taken by the CC and defines relevant terminology [24]. Part 2 [3] defines a set of *Security Functional*

Requirements that resemble the security controls defined in the SP800-53. Parts 3 [25] focuses on *Security Assurance Requirements* and the methodology that must be used for evaluating IT security. Our work focuses explicitly on Part 2, designated as ISO-15408 or simply referred to as *CC-Part2* in this paper, as it describes the security requirements organizations must follow. The CC-Part2 follows a three level structure, like the SP800-53, but is organized using a component-based format. At the top is the *functional class* which describes the category, such as security audit. Next are a set of *functional families* that denote general security requirements, e.g., the need for audit storage protection. Below each of these are one or more *components* which describe particular details of the security requirement and may be hierarchical to each other.

2.2 Security requirement extraction and modeling

Before discussing how security requirements can be extracted and modeled, it is important to define certain terminology in order to proceed with a common vernacular. Foundational work by Haley et al. [26, 27] provides a formal treatise on the security requirements engineering process and complements the NIST security life cycle [2, 13]. Haley defines *requirements* in terms of security goals and threats to identified assets. *Assets* are "objects with a direct or indirect value" [27] which if lost would incur some harm. A *threat description* describes the harm caused if an action is performed on an asset. *Security goals* are threat descriptions prefaced with "prevent." Finally, *security requirements* are operationalized security goals that constrain the system design to prevent or reduce possibly harms to the set of identified assets. The NIST SP800-53, like the CC, defines *security controls* [2, 3] as "safeguards" or "countermeasures" that when implemented, provide a "level of security due diligence" for organizations. Contrasting this definition with Haley's, we can say that security controls contain one or more security requirements that constrain system design to prevent harm to identified assets, i.e., they fulfill the security goals of the organization.

In addition to this defined core terminology, the security requirements extraction process produces *context-specific terminology* [2], such as what the definition of an auditable event is or what organizational parameters mean. *Ontologies* [28–33] are typically used to organize and structure context-specific terminology, which helps security analysts to classify and reason over security controls in regulatory documents. Ontologies, at their core, consist of a set of *entities* [34] that represent concrete concepts, or types of things, and a set of *relations* [34] that connect disparate concepts. For instance, a few entities might be John Lennon, Person, Beatles. These entities could be related together using common ontological relationships like *is_a* or *part_of*, e.g., John Lennon *is_a* Person who is *part_of* the Beatles. Applications of ontologies to the security lifecycle include assisting organizations in the security control selection and design-time software engineering process [33, 35], providing support during the policy decision making process [36], enabling run-time adaptations (such as web service replacement) [37], and providing the building blocks for structuring regulatory terms and controls [38, 39].

In the latter realm of classifying terms and security controls, Lee, et al. [38, 40] examined the DITSCAP [21], the predecessor to the DIACAP [1], and its family of documents. Their goal was to facilitate security certification by linking together federal regulatory requirements across document abstraction levels. They defined three requirement levels to separate high- and low-level security requirements [38]. At the top were *generic requirement* document categories, where the OMB A-130 [23] fit, that described the "Why" of the security plan, security development strategy, or technical control summary. Drilling down yielded *domain spanning requirements,* embedded in policy documents such as the DoDI 8500.1 [22], which expanded the higher-level categories. Finally, at the bottom were *sub-domain requirements* which further specified "how" the higher requirements should be implemented in systems, using controls in the DoDI 8500.2 and operational guidelines in the DoD 5200.2 [20].

Lee et al. [38] provided basic natural language (NL) requirement extraction by (1) decomposing complex requirements into atomic statements, (2) determining the level of abstraction that an atomic statement applies to, (3) assigning it to one of the DITSCAP requirement categories, and (4) eliminating any conflicts that may exist between documents. Several ontological linking relations, such as *comply_to*, *specific_to*, and *realized_by*, were defined to link the requirements between levels. The end result was an informal, natural language, network of requirements that could be used to build a DITSCAP checklist for organizations to assess compliance against. While helpful and novel when originally developed, their approach is limited by its informal nature and by its attachment to DITSCAP-related documents, which are now obsolete.

Tsoumas et al. [39] develop a risk analysis approach that relates various elements involved in the risk assessment, mitigation, and certification lifecycle. Their framework allows high-level policy statements (the "what") to be linked to lower-level security controls (the "how"), similarly to Lee's approach. In their work, a system profile is constructed to identify system assets and represent the security controls used to protect them. This construction relies on statements in OWL-DL [34], which are quantified over the assets in order to define the necessary and sufficient conditions required for policy compliance. OWL, or the *Web Ontology Language* [34], comes with a RDF, i.e., *resource*

description framework, serialization that facilitates common interchange. Despite OWLs wide acceptance as a W3C standard [34], it can be unnecessarily complex and is not the right representation tool for every situation.

Taguchi et al. [8] construct a framework, targeted at the Common Criteria, for modeling security requirements and facilitating security assurance. Their framework consists of a *meta-model* and a multistep requirements and assurance lifecycle for specifying and verifying use and misuse cases. The meta-model, constructed similarly to ontologies, defines the structure of the Common Criteria using a two-actor environment (a user and threat agent) and set of use cases and security functions that realize CC security functional requirements and prevent threats to assets [8]. The phases of their lifecycle resemble the NIST lifecycle [2, 13] and assist security analysts by mapping security control requirements in the Common Criteria to use/misuse cases for certification efforts.

Breaux and Anton [19], in widely accepted work focused on healthcare regulations [19, 41], develop the notion of using *semantic patterns* for extracting and expressing privacy policy requirements from HIPAA regulation statements. Although the concept of patterning natural language is not new, their approach successfully uses it to develop triples of actor, action, and object to state compliance constraints as *exceptions*, *obligations*, and *rights*. Despite the exceptional nature of their work, their patterned approach is specific to HIPAA because the privacy statements follow a regular format that is not found in, or applicable to, security control regulatory documents [19]. Privacy controls, which have been forcibly separated in the NIST SP800-53r4, can be generalized into handling, storing, and transmitting. HIPAA has a very strong focus on data handling by personnel and is only related to health data, which is why Anthem Blue Cross was not penalized when hackers were able to access 80 million US personal identifying information, including social security number, birth dates, and employers, but not health care information [42]. The policy statements are at a higher level than the NIST and related security controls. Thus, they are not constructed with a structure that would be amenable to formalization and this is one of the drawbacks of HIPAA. Others using semantic patterns include Daramola et al. [43], who create what they refer to as *requirements boilerplates* that allow requirement engineers to fill in the blanks with parameterized information related to their application(s). Example boilerplates include semantic patterns such as "The ⟨system⟩ shall be able to ⟨action⟩ ⟨entity⟩" [43].

Several tools and research efforts have tried to provide a more generalized patterning process for examining functional requirements (not necessarily security requirements), expressing them in a patterned way, and using the patterned content to support the derivation of class elements. For instance, the Requirements Analysis Tool (RAT) [44] adds structure to NL statements taken from arbitrary software requirements documents to develop a set of heuristics, e.g., patterns, that enable the construction of a high-level system class diagram. Another tool, called Circe [45], constructs and uses domain-specific glossaries, which contain a list of key terms and synonyms found in NL requirement document text, to perform a series of canonization and tokenization steps to transform plain NL text into a structured format. It uses model-action-substitution rules (*MAS-rules)* in the transformation process, where a single MAS-rule is a triple, $\langle m, a, s \rangle$ that applies to a requirement t. When m matches a fragment of t, the action a is applied resulting in the fragment of t being replaced by s. The end goal is a model of the extracted requirements which experts can evaluate for correctness.

2.3 Formal system modeling

Once requirements have been extracted from regulatory documents, system designers must be able to demonstrate their systems satisfy the requirements. Research [26, 37, 39, 46] has shown that ad hoc approaches that do not formally represent security features during the system architectural design process are much more prone to vulnerabilities and, thus, suffer from increased risk of exploitation. The challenge for system designers is to represent their systems in a form amenable to certification against the extracted regulatory requirements. Secure cloud system modeling relies on demonstrating *virtual isolation* [47], *security property preservation* in web compositions [48, 49], and *operational correctness* [7, 47, 50–53]. *Virtual isolation* ensures that each cloud execution environment is accessible only be approved parties [47], e.g., one cloud client cannot view or modify data or functionality being used by a second cloud client and vice versa. Preserving *security properties* [48, 49] requires demonstrating that there are no *local* (i.e., pairwise between services) or *global* (i.e., end-to-end across service compositions) violations of data integrity, confidentiality, or availability regulatory requirements. Such violations may arise as a result of trust [49, 54] or communication [48, 49, 54] issues that may exist between the selected web services. The last issue, *operational correctness* [47], refers to systems or compositions of services performing as expected. In other words, functionality meets its stated goals without introducing error scenarios that could be potentially exploitable.

Overcoming these security modeling challenges has been the focus of numerous works [46–49, 55, 56]. Among those works, Bleikertz et al. [47] follow a graph theoretic approach to express virtual isolation and certain operational correctness requirements as graph rules, e.g., for isolation two nodes outside of the same security domain may not communicate. Their graph rules are then mapped onto the formal language VALID [47] which uses model checking to

compare the run-time states of virtual machines (VMs) in the cloud against the defined security graph rules. Singaravelu et al. [49] examines the access control, encryption, and trust implications associated with end-to-end messaging in web service compositions. They introduce WS-FESec as a WS-Security extension that improves the end-to-end handling of confidentiality and integrity constraints in messages [49]. She et al. [48] propose a message exchange protocol for information flow control among composed services to reduce violations. Their *carry-along policy* and *pass-on certificate* advertise service flow control policies and prevent information leakage by providing appropriate credentials, but it incurs a large overhead due to the increased interaction among the services required to appropriately process the policy directives.

Cloud systems can also be modeled using a *coordination language* approach. Coordination languages have been used successfully to formally verify operational correctness [50, 51] and security properties over static SOA specifications [7, 51, 57]. The primary coordination language construct is a *tuple space*. Tuple spaces are a well-studied data structure [7, 57–59] that facilitate a data-driven model of component interactions. Tuple space usage originated with the parallel programming language Linda [59], which provided the three operators *in*, *out*, and *rd* to allow tuples to be, respectively, entered, removed, or read from the tuple space. Merrick et al. [58] established a set of scoping rules that allowed Linda tuple spaces to be nested or isolated from one another. KLAIM [51] first introduced permissions to tuple spaces providing access control mechanisms capable of limiting which processes or components in a system can manipulate or read the content of tuples in a space. Their approach relies on a set of static access control policies to limit certain processes to their respective spaces and is course grained meaning single tuples and data fields cannot be restricted within a tuple space.

Recently, Linda-like tuple spaces have been applied by Bravetti et al. [52] to model and secure coordination in untrusted interaction environments that occur between composed services. Their work developed a language called *SecSpaces* [52] which refines Linda tuples with asymmetric key-based *control fields* that provide fine-grained access control over elements in a tuple. A SecSpace tuple may only be read, i.e., *rd*, if the provided credential matches the stored private key. However, despite these advances from previous Linda iterations, *SecSpaces* lacks the formal proof logic necessary to prove temporal interaction properties. Another coordination language, named X-UNITY [7, 53], pronounced "cross-unity," offers all of the fine-grained access control and encryption capabilities of *SecSpaces*, but also brings to bear a powerful temporal proof logic associated with its predecessors [50, 60]. Of the coordination languages available, it provides the most fertile ground for representing services in the cloud and the best proof theory for proving temporal properties over service specifications. The *compliance models* derived by our extraction process can be used to direct certifications in any of the above, and many other, temporal logic-based modeling paradigms.

2.4 Addressing security requirements during development

Extracting and modeling security requirements strictly for use during the design phase of the software development lifecycle is not sufficient. Instead, those requirements should also be applicable throughout the implementation, certification, and maintenance phases. Multiple research efforts [61–68] have explored how formalized security requirements and formal methods used in the design process can be translated into developer-friendly tools for use during product implementation. Generally, these approaches focus on providing traceability from requirements to code to test cases [61–63], weakness, threat, and vulnerability repositories knowledge re-use for risk assessment and mitigation [64–66], and tool support for certification [67, 68].

Keeping code traceable to the requirements that generated it to the test cases that assess whether those requirements are satisfied or not is the central focus of many efforts within the requirements engineering community. Wang et al. [61] examined this problem in the context of open source projects using a mapping approach called a *requirements traceability matrix*. Using an open source project called iTrust as a case study, they identify three common situations that lead to traceability concerns—*extending an existing requirement*, *implementing a requirement*, and *realizing a previously unfulfilled requirement*. Ghezzi et al. [62] focus on formal methods that account for the iterative effect of continuous evolution. They use an incremental model-checking approach based on evolving statecharts to state logical properties and compare them against a system model during agile development. Mahmoud and Niu [63] focus on refactoring techniques that recover structural vocabulary terms to improve information retrieval (IR)-based traceability techniques that span code, tests, and requirements. Their approach works by making textual artifacts more normative through systematic refactoring using three operations (Rename Identifier, Move Method, eXtract Method) adapted from other in IR-based approaches. Rename focuses on ensuring terminological consistency across software evolutions in a way that maps to the original design requirements. Move method removes misplaced signs of traceability by ensuring that code is packaged in the place it was designed to be in. Finally, extract method removes duplicated code where requirements may be traced to multiple instances of the same code, based on copy-paste style programming.

Knowledge re-use efforts largely focus on using formal methods and requirements side-by-side with repositories, such as CWE [15] and CVE [16], to generate certification criteria for use during design and development. One approach by Hermoye et al. [64, 65] uses extends formal semantics in the KAOS [69] framework to allow for the use of attack pattern libraries and associated countermeasures. Attack patterns express anti-goals, domain properties, and predicates generically. The re-use approach is 4-phased. First retrieve relevant generic attack patterns. Next, specialize them, by adapting the patterned concepts to fit the specific cases applicable for the system of interest. Third contextualize anti-goals according to the *how* and *why* the specialized attack applies to the system. Finally, derive new or re-use existing requirements that act as countermeasures for the identified contextualized threats. In similar work by Saeki and Kaiya [66], the Common Criteria is used alongside the ECMA-271 E-COFC [70] as the source of knowledge for requirements elicitation. As with Hermoye, Saeki and Kaiya [66] focus on the role of a threat catalog, however, the countermeasures that they emphasize (i.e., the security requirements that mitigate identified attack patterns) are derived directly from security controls specifying Security Functional Components in the CC.

Tool support is another area focused on developers. It is important for approaches relying on formal methods and formally stated requirements to consider how those methods and requirements can be presented developers in an understandable and usable way [67]. Hence, tools are often developed to assist developers in applying formal requirements analysis techniques during development. Yu et al. [68] propose a meta-model and associated automated tool, called OpenRISA, that makes the connection from formal methods to formal arguments that can be used in support of the risk assessment process. Their work extends earlier security argumentation work by Haley et al. by adding support for public security catalogs (such as CAPEC [71] and CWE [15]) and automatic reasoning. Figure 4, from [68], overviews the RISA approach. The process seeks to prioritize risks and identify mitigations using a combination of the security requirements on the system and public repositories of attack patterns and software weaknesses. Our work in this paper fits within steps 2 and 3 of the RISA approach, as we have highlighted in red atop the figure. Our work offers increased precision for security requirements as well as traceability to the regulatory documents where the security functional requirements were generated. The logic-based syntax of our requirements process is integrable with approaches such as the RISA method.

3 Governance patterns and formalization

The first step in extract requirements from regulatory documents involves identifying common elements expressed in the natural language text of the document. This section defines a set of *security governance patterns* capable of directing control statement property extractions. Associated

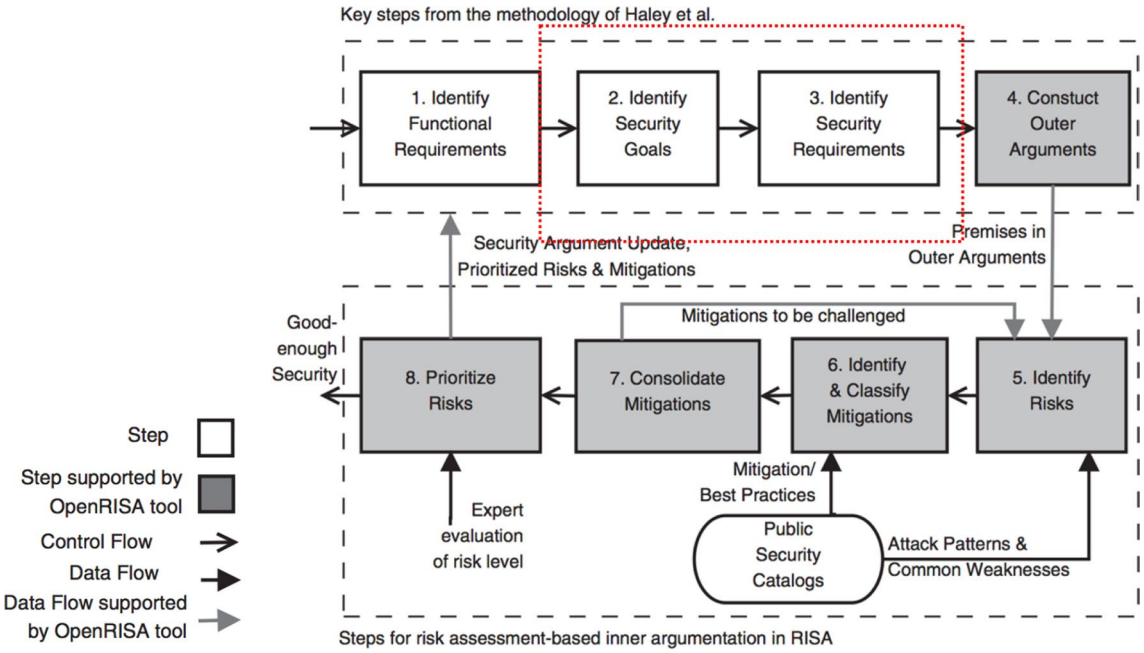

Fig. 4 RISA method from [68]. Improving steps 2–3 (overlaid in red) through more precise and integrative requirement specification, as in the compliance hierarchy modeling process, are within the scope and purvey of our work in this paper (color figure online)

with each pattern is a *modeling template* that maps extracted control content to a temporal logic construct based on the pattern type. The logic constructs are reminiscent of *first order logic*, but also rely on elements of *temporal state-based logic*, such as in [60]. The patterns were empirically derived by examining existing controls in existing regulatory standards in a way that allows them to be applied regardless of organizational, business, or government agency boundaries. To ensure future re-use, the derivation process of each pattern is clearly defined. This means that compliance predicates formed by applying the patterns can be used for verifying an information system's behaviors, without being tied to specific document syntax. It also means that the pattern-based extraction and modeling process can accommodate newly introduced controls as they come out, such as those that may specifically address web services and cloud security, in a way that can be transformed to a verification construct consistent with other existing security constraints. It goes beyond the NIST SP800-53Ar4, which provides insight into using "examination," "interview," and/or "testing" to establish parts of security controls, by highlighting relationships among controls so that the certification techniques can also be directly related depending on the family and individual specification of the control.

All security controls have a general security policy *intent*, or area of security within an information system, such as separation of duties or cryptographic mechanisms. Determining the intent is the first step in the patterning process and is based on the placement of the control in the governing document, its descriptive title, and the context of its goal statement. Assigning intent to security controls forms an *intent class* that relates controls based on their specific requirements. If an examined control does not fit into any existing class, then a new class is formed. For example, patterning a control related to determining which events should be audited would introduce an "auditable event" intent class. Another control, perhaps describing the type of encryption mechanism for audit records, does not conceptually fit with types of auditable events. Instead a new class with the intent of "audit protection" would be created. Controls in classes that are closely related can be periodically reexamined as the extraction process progresses to determine if any can be deemed a better fit in another intent class. The result is a partitioning of controls at a fine enough granularity to separate concerns, but not so fine that there are unnecessary hierarchies.

Each intent class will ultimately form a separate security control hierarchy. Each control in a class must first be extracted by a governance pattern. The pattern used is selected based the control's information content. Analysis of the governing documents indicates that a security control targets one of three information system functional goals (1) it requires the information system(s) to abide by the formatting and parameters that the organization *imposes*, (2) it supplies information about how a function should *perform* some mission critical security functionality, or (3) it defines mechanisms that *protect* critical assets. Other technical controls relate to the definition of policy artifacts that the organization *designates* for human review and intervention. Though this last type is a pattern, it is non-technical and thus outside the scope of this work, which specifically focuses on technical requirements.

Each governance pattern, i.e., *imposes*, *performs*, or *protects*, requires security analysts to select a certain *level* that denotes the scope of the control. The possible levels include *Org* (organization), meaning the control statement is based on an organization-wide directive, *InfoSys* (information system), meaning it applies to an information system implementation, or *Comp* (component) meaning the statement applies to a specific component or process within an information system.

Using a box notation similar to that found in Z [72], we define a *model template* to house and formally express the information in each patterned control. The template, shown in Fig. 5, consists of three fields. The top identifies the control statement by its document of origin and identifier. The middle is used to formally define a set of *declared variables*, extracted by the security control pattern. Unless specifically designated, these variables are universally quantified within the instantiated template. The bottom is the constraint specification that defines logical relations embodying compliance requirements in the extracted control. Requirements that express a temporal state change are represented using a "leads-to" relation. The statement p "leads to" q means that when a system reaches a state which satisfies the predicate p, eventually it will reach a state satisfying another predicate q, whether or not p still holds in q [35]. Bolded text in a template denotes a label that may be re-used across controls. These labels are helpful when multiple controls specifically reference elements defined by other related controls. All patterns use the same template, but may have slight variations in their actual instantiation and expression based on the information extracted by the governance pattern.

Applying the patterns to the full set of organizationally selected controls results in an enumeration of compliance requirements, that when grouped into compliance predicates, collectively define the organizations mandated security

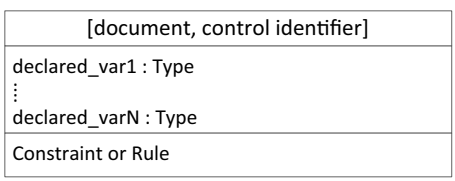

Fig. 5 Modeling template

profile. The following sections detail each pattern, describing the types of controls it applies to, what information is to be extracted, and how to formalize the extracted information. The patterns are highlighted in the context of the running audit case study.

3.1 Imposes

Security control documentation requires organizations to select certain policy parameters and impose them on designated critical information assets. To accommodate and represent this type of security control, we created the *imposes* pattern. The imposes pattern specifically applies at the *Org* level meaning that the organization applies policy parameters to certain identified assets in all of the information systems it controls and uses. It is important to note that imposes cannot apply to *Info Sys* or *Comp* levels. The *imposes* pattern is read as follows:

Org *imposes* ⟨policy entities⟩ *on* ⟨assets⟩

where ⟨policy entities⟩ are the organizationally defined parameters that constrain information assets denoted by ⟨assets⟩ or, more generally, all information systems (when ⟨assets⟩ is left null). Policy entities govern particular facets of the system such as specifying the encryption method systems must use, establishing information record keeping requirements for audit record assets, or defining auditable events. For imposes patterned controls, the definition and representation of policy entities constitute the compliance requirements associated with the control statement.

Formulating an *imposes* requirement involves identifying the policy entities and the assets they are imposed on and then mapping the extracted fields to the modeling template from Fig. 5. This mapping includes typing, quantifying, and perhaps constraining each extracted control parameter. When typing control parameters, we assume a basic set of *primitive types* exist, such as String, Integer, Boolean, and Component. Other *complex types* may be defined during control extraction to represent the specific semantics of control parameters. Complex types are generally *objects* that are developed from primitive types. An example might be a type called UID, which is a semantically meaningful type of Integer that defines a user ID. The typing process also allows for the definition of sets, simply denoted as *Set* ⟨type⟩, and powersets denoted as $\wp$ ⟨type⟩, where type is a defined primitive or complex type. Here, a set is an unordered collection of the underlying type and a powerset is a set of sets of the underlying type. Finally, the notation ↑ ⟨typename⟩ is used to designate a *Compliance Type* which is a compliance predicate that results from the development of a semantic hierarchy elsewhere in the compliance model; this is discussed extensively in Sect. 4.5.

As an example, Fig. 6 illustrates an audit security control from the Common Criteria called FAU_GEN.1.2. This control, from the functional audit family, imposes six content parameters on all information system audit records and allows for additional content to be included in the record. During control extraction, an intent class called *Audit Records* is created to collect groups of controls, including FAU_GEN.1.2, that are related to how an audit record is defined. The instantiation of the *imposes* pattern for FAU_GEN.1.2 appears under its NL control statement in Fig. 6. The filled modeling template to the right identifies the Common Criteria as the document and FAU_GEN.1.2 as the control id. Below the title box are the declared variables which represent the policy entity parameters extracted by imposes. Given these parameters, the information asset, called **record,** is defined to be "at least" the required parameters, as denoted by the symbol ($\asymp$), but could potentially have additional content parameters. The record will contribute to the AuditRecord type as shown later in Fig. 16. A discussion describing how FAU_GEN.1.2 fits into the *Audit Record* compliance hierarchy is provided in Sect. 4.

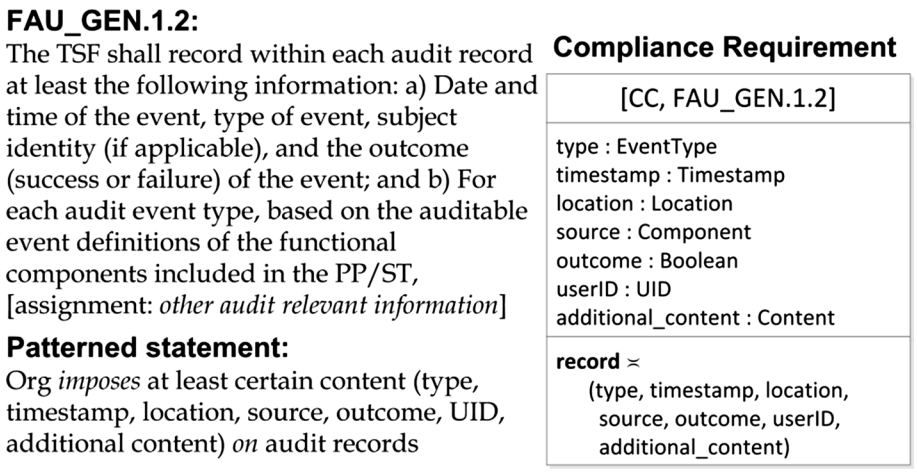

Fig. 6 Imposes pattern applied to the Common Criteria control FAU_GEN.1.2

3.2 Performs

The second pattern, called *performs*, expresses constraints that relate directly to properties of system functionality. The pattern relies on an *action* to specify the functional behavior, a set of *pre-conditions* and *input* that constrain the application of the action, and a set of *post-conditions* that detail the results of the action's effects on the system. Often a post-condition for a security functional requirement dictates that an *asset* or *artifact* is created or modified. For this reason, the pattern explicitly separates generic post-conditions from assets or artifacts which must be produced as a result of the execution of an action. For policy statements that do not specify asset or artifact creation, this field should be left null. Performs is thus defined as follows:

⟨level⟩ *performs* ⟨action⟩
given ⟨preconditions⟩ and ⟨input⟩
that results in ⟨postconditions⟩

where ⟨level⟩ is either *InfoSys* or *Comp* (cannot be *Org*). Often the line demarcating the *InfoSys* and *Comp* is blurred, since the information system may designate a certain component to perform the action. In such cases, *InfoSys* is chosen as the level for consistency and the use of a component becomes part of the constraint in the model template.

Declared variables in a *performs* pattern can be input and output variables, and relations between variables, such as functional mappings. One distinction is the use of the temporal property *leads-to* (⊫▷), borrowing from UNITY semantics [60], in the constraint specification to denote a *progress property* in which eventually a state will be reached from the pre-conditions (which may be simply *true*) where the post-conditions are true. Sometimes, the pre-conditions and post-conditions involved in *performs* are not explicitly stated in the security control. In such cases, the pattern instantiation relies on the semantics of the action to imply the conditions as in the case of FAU_GEN.1.1(c) shown in Fig. 7, where the phrase "generate an audit record" implies that the record does not already exist. Additionally, the *input* expressed may reference an abstract information system architecture.

Figure 7 shows the extracted performs pattern statement below the stated natural language (NL) control FAU_GEN.1.1(c) from the Common Criteria. A component-based implementation of the information system is inferred given the foundation for the Common Criteria specifications. An inference is also made that the Target of Evaluation Security Functionality (TSF) does not generate audit records for all components. Therefore, "a component" is added to the *given* section of the pattern. This inference is based on other similar controls from other documents. For compliance verification, an organization could select all components to be audited, in which case the *performs* constraint would be used to reason over each one separately. The *Compliance Types* ↑AuditableEvent and ↑AuditRecord are used within the control, though they are defined as part of other intent classes as shown later in Fig. 16. In the model template (right side of Fig. 7), the input parameters and a function, called *Gen_Rec*, describing audit record generation actions, become declared variables. A leads-to property (bottom of template on the right side of Fig. 7) expresses that for any component, *a state in which an auditable event occurs in a selected component and a corresponding record has not been generated for the event, eventually leads to a state in which the generated audit record for the event is placed in the component's audit log*. Note that *Gen_Rec* does not have to be part of component *c*. Neither does the evaluation of an occurrence of the event. Also, the notation (') is used to denote the "after state" of a variable as in c.auditLog and c.auditLog'. Stated specifically, a variable *x*' denotes the "after state" of a variable *x*. Since the *performs* pattern denotes a requirement for a state change, its assessment could be in the form of evaluating traces, such as the use of security hyperproperties [73].

3.3 Protects

A third pattern is used when a security control identifies assets, processes, or functions that must be protected against

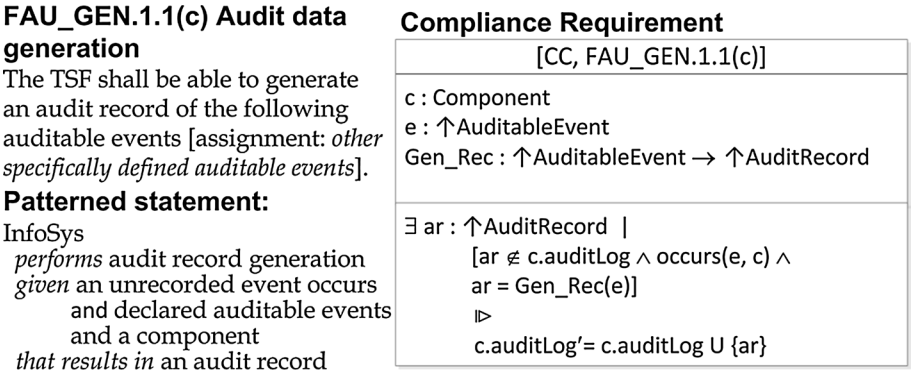

Fig. 7 The performs pattern as applied to the Common Criteria control FAU_GEN.1.1(c)

some negative activity, such as malicious user actions. This security control type prescribes a selection of high- or low-level protection mechanisms. For technical controls, these protection mechanisms can include *performs* constraints, thus incorporating that pattern. Given these considerations, the *protects* pattern captures the relevant information as follows:

⟨level⟩ *protects* ⟨asset⟩
using ⟨mechanism⟩
to prevent ⟨activity⟩

where ⟨level⟩ can be *InfoSys* or *Comp*. *InfoSys* more generally represents the need to protect the asset using some *performs* function specified in the system, while *Comp* indicates an actual component which deploys the mechanism responsible for protecting the asset. The protection ⟨mechanism⟩ is imposed by the organization and may be either a type of encryption method, digital signature, or monitoring process depending on whether the ⟨activity⟩ violates a confidentiality, integrity, or availability security objective, respectively.

The *protects* pattern may need additional context to extract the ⟨mechanism⟩ from the control text. The mechanisms may appear elsewhere in the control document such as the NIST *supplemental guidance* section of a security control [2] or may be embedded in a separate security control that has a semantic link to the control being expressed (discussed further in Sect. 4). *Protects* involves declaring and quantifying the assets for protection, the activity to be prevented, and the function or set of functions to be used. Each function is expressed as part of a *performs* statement, which must be included or semantically linked to the control that instantiates the *protects* pattern. Function labels, shown in bold, label *performs* constraints. These labels are referenced within the *protects* instantiation of the control statement to indicate the logical dependencies needed to describe the mechanism function expectations.

Figure 8 shows the *performs* constraints for AU-10.E3 that defines **review(r)**, for an audit record *r*. The **review(r)** function is one of four mechanisms used in AU-10.G to prevent repudiation of audit records. The control text establishes the need for a reviewer's UID to be stored in the audit record upon review and requires the UID to be signed. The *performs* pattern is applied to the control text on the top left of Fig. 8 and results in the pattern instantiation below it. Contextually, the "information" described by the control text refers to all audit records created as the result of a user's actions that have been previously validated to insure the user UID has not been modified. A binding mechanism (*BindRev*), such as a digital signature mechanism, is assumed as input. This definition of **review(r)** states that whenever an audit record, *r*, has been validated (i.e., using **validatedUID(r)**) and a review request is initiated (*occurs(e)*), a state eventually results that adds the reviewer's UID to the audit record ($r' = bindRev(r, rev\_uid)$) and signs it ($r'.revUID.signed$). The concept of signing, although not directly in the control statement, is inferred from the control's references to "credentials" and "associates the identity."

AU-10.G incorporates AU-10.E3, and three other controls, as part of a compliance requirement to ensure non-repudiation of audit records. Figure 9 shows the result of extracting the requirements of AU-10.G and instantiating a *protects* pattern. The context of the control implies that the set of all audit records is the ⟨asset⟩ in the pattern. The control does not directly specify the mechanism to protect the audit records because it relies on the control enhancements which are referred to by their labels (including **review(r)** as developed in Fig. 8.

The ⟨activity⟩ the control seeks protection from is repudiation. Thus, *repudiation* is of type *Activity*. The function *prevents* is standardized within the *protects* pattern to collect and apply the mechanisms on the assets. In this case, *prevents*(*repudiation, r*), in Fig. 9, means the activity of *repudiation* as applied to the set of audit records *r* is prevented from occurring as long as at least

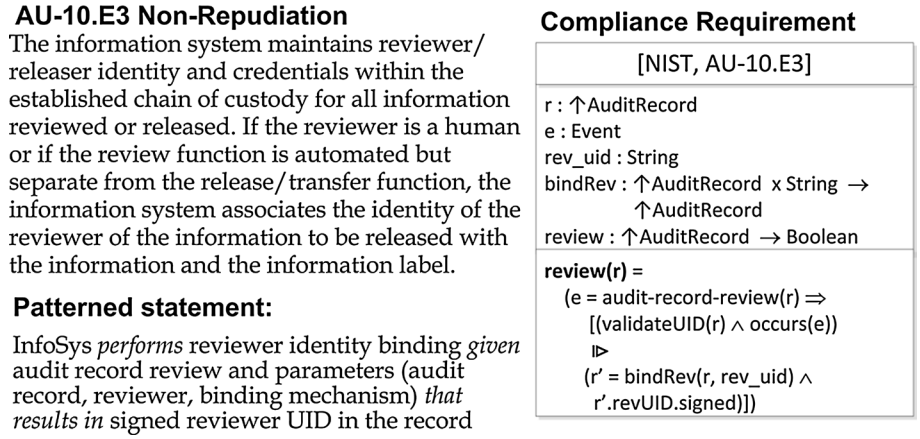

Fig. 8 The performs pattern within AU-10.E3

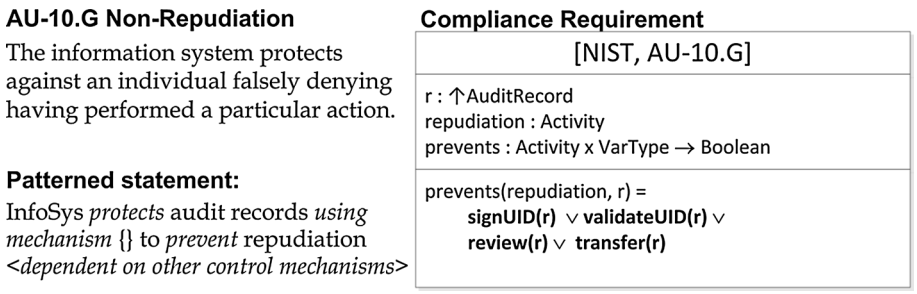

Fig. 9 Protects pattern applied to NIST control AU-10.G

one of the mechanisms (e.g., **signUID**, **validateUID**, **review**, and **transfer**) is performed.

In the case of AU.10.G, *protects* expresses the overall mechanisms to prevent a bad state from occurring while at the same time dictating the state changes allowed by the mechanisms. The **signUID**, **validateUID**, **review**, and **transfer** mechanisms are individually instantiated as *performs* compliance requirements and hierarchically connected to AU-10.G via a semantic relation (discussed later in Sect. 4.2). As long as the *performs* compliance requirements are satisfied, *prevents*(*a*) holds and repudiation cannot occur in the information system. This knowledge, required for formalizing a *protects* property, is derived from the semantic relations connected to AU-10.G. Understanding these relations is the topic of the next section.

4 Creating semantic hierarchies using semantic relations

Determining the compliance predicates that provide coverage of the security controls requires identifying the relationships that exist between security controls, starting with those in the same intent classes. As these connections are investigated, within each class, a *semantic hierarchy* emerges that identifies a *dominant control,* i.e., a control whose expression encompasses the requirements of the class. This section defines four *semantic relations* based on their structural properties and the subsequent representations that describe how compliance requirements are affected by, and propagate over, the relation. The relations are irreflexive, anti-symmetric, and transitive such that for any relation type (except *forms*), if a control *C1* is related to another control *C2* by a relation *R1* and *C2* is related to a third control *C3* by relation *R2*, then *C1* is related to *C3* by *R1*. If there is a relation *R3*, such that controls are related by *R1*, then *R3*, then *R2*, transitivity is not proven to hold in all cases. The audit case study is continued for clarity and illustrative purposes.

4.1 SubsumedBy

A control, *c1*, is *subsumedBy* another control, *c2*, as represented by $c1 \rightarrow c2$ iff:

- both controls are instantiated by the same pattern and
- the specification constraint of *c2* implies the specification constraint of *c1*.

Figure 10 exemplifies three controls, ECTP-1, FAU_STG.1.1, and APP3690.4 with intent *Audit Protection* that are *subsumedBy* AU-9.G, with the same intent. AU-9.G specifies the need to protect all audit records, the audit trail, and all audit reports from unauthorized access, modification, or deletion, thus subsuming the types declared for *assets* as well as the activities being prevented on those assets. No specific process is required by the control constraints, so organizations may construct any protective mechanisms to ensure that *prevents* is satisfied.

4.2 UsedBy

Figure 9 illustrated the instantiation of a *protects* pattern with labeled mechanisms, one of which, i.e., **review(r)**, was described in Fig. 8. The *usedBy* semantic relation, represented by $c1 \twoheadrightarrow c2$ in Fig. 11, indicates the inclusion of a predicate *s* in *c1* (in this case a labeled *performs* statement) in a predicate *t* in *c2,* such as the *prevents* repudiation activity. Parameters for *s* and *t* can be used within the label to clarify which predicates in *c1* are *usedBy* predicates in *c2*, if it is not obvious.

Examining Fig. 11 in more detail shows that FAU_GEN.2.1 specifies and labels the *performs* statement, **signUID(r)**, which takes an audit record *r* and eventually returns it as a signed record *r'*, using the organizationally defined digital signature mechanism represented by *signMech*. Though no formal relationship exists in the documents, FAU_GEN.2.1 (Common Criteria) can be linked to the NIST AU-10.G and AU-10.E2 using *usedBy* to provide a functional requirement detail regarding signing that is missing in the NIST. AU-10.E2 defines a validation mechanism, *validateUID*(*r*) that examines an audit

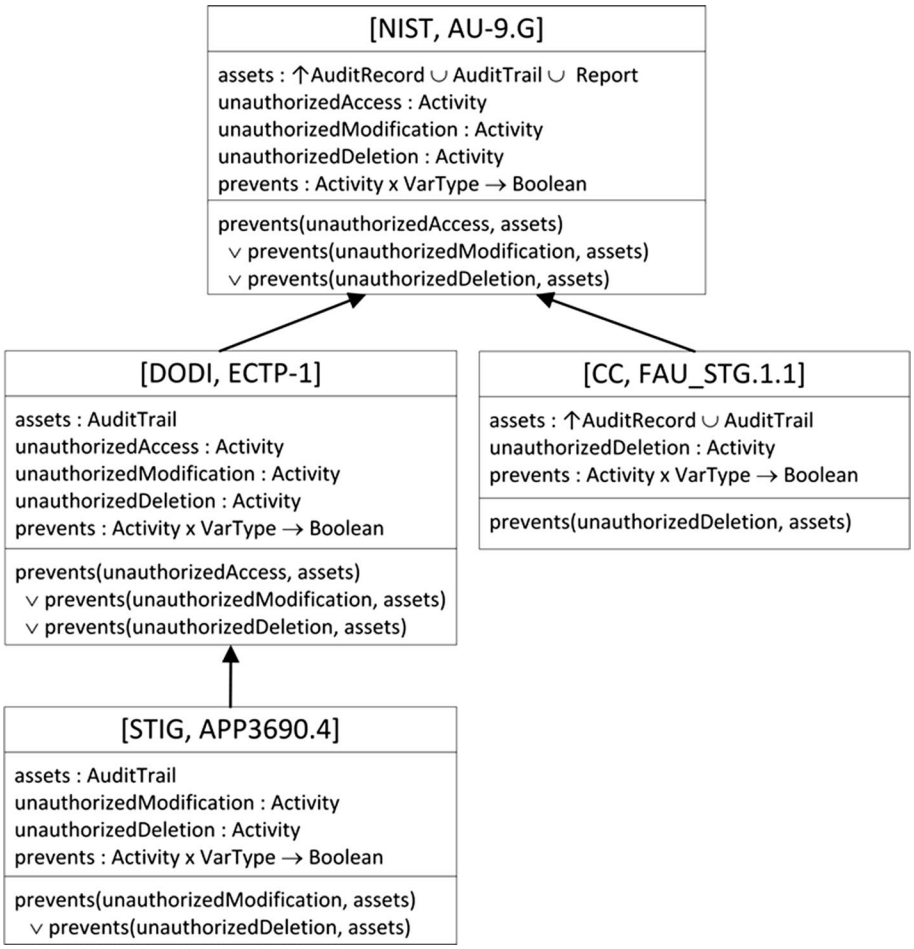

Fig. 10 Portion of the audit protection control hierarchy that is subsumedBy AU-9.G

record for a proper digital signature whenever a review request is made on the record. If the record is signed (i.e., **signUID(r)** is true), then eventually a state is reached when the record is verified, using the organizationally defined *verify* method that supplements *signMech*, allowing it to be reviewed.

AU-10.E3 (shown originally in Fig. 8) expresses the binding of a reviewer's user ID to the record ensuring the reviewer cannot repudiate the review. AU-10.E4 in Fig. 11 states that the audit record reviews are validated prior to transferring the record outside the security enclave. Thus, **transfer(r)** is true if an audit record transfer request occurs that eventually results in a state in which *verifyRevUID* is performed on *r* to ensure that the signed reviewer *uid* is legitimate. Each of the predicates **signUID**, **validatedUID**, **review**, and **transfer** prevents repudiation and therefore are *usedBy* (and included in) the *prevents(repudiation)* predicate detailed in the constraint portion of AU-10.G. Because the predicates are also included in the other non-repudiation mechanisms, *usedBy* denotes that relationship as well.

4.3 Structures

Information systems generally require a number of security-sensitive assets and types. Individual security controls related to assets and types may impose a representational format as well as define asset and type attributes. An example in Fig. 12 is the STIG APP3620 control that requires a secrecy level attribute that is one of {*top secret, secret, unclassified*} to be associated with all auditable events. Thus, the control defines a specific structural element of the auditable event definition.

In fact, a type's structure, or format, may be partially defined in multiple security controls to provide a fuller representation of the type. The *structures* semantic relation appends type definitions to assets defined in other controls by denoting specific formatting elements. Thus, *c1 structures c2,* represented by *c1* $\rightsquigarrow$ *c2,* states that a declared variable in *c1* defines some portion of the structure of a declared variable in *c2*. Parameters of *structures* identify the appropriate variables in *c1* and *c2*. If only one variable is declared in the schema for the control, then that variable

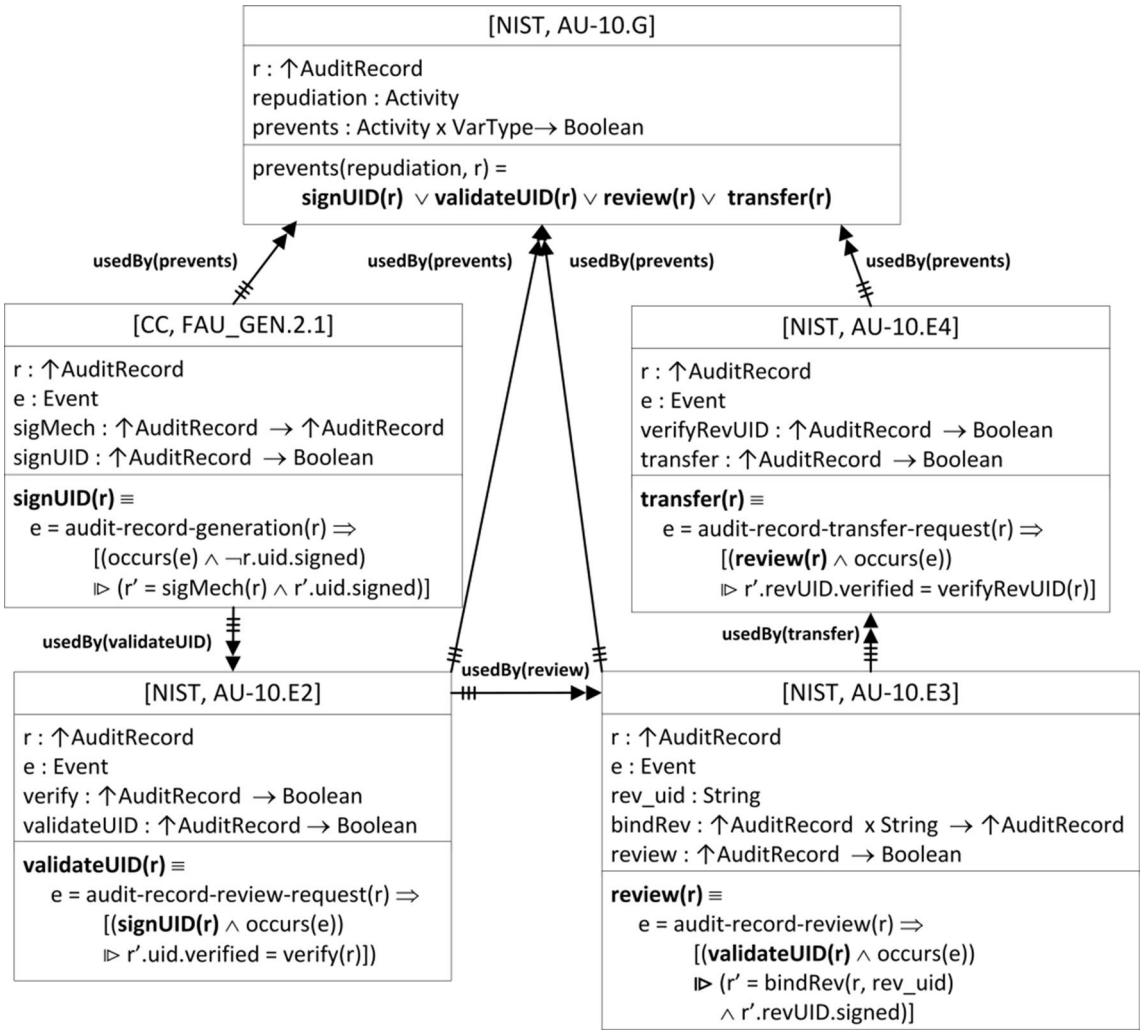

Fig. 11 The non-repudiation control hierarchy with the usedBy relation

is a parameter by default and is not made explicit within the semantic relation.

Figure 12 illustrates two scenarios using *structures*—when it is applied to an asset type and when it is inserted as an element of a tuple to format a type. Control FAU_SEL.1.1 specifies the various elements of an auditable event. As used in Fig. 6, the $\asymp$ symbol denotes that an **event** *is at least* composed of the elements specified in the associated tuple. This constraint is transitively passed on to AE, in AU-2.G, which defines all possible auditable events. Effectively, the relation between FAU_SEL.1.1 and AU-2.G, i.e., **structures(event, AE),** expresses that all auditable events at least contain the information in the **event** tuple. This example corresponds to the first case described by the **structures** relation definition. **Structures** is also used as a relation between APP3620/FAU_GEN.1.1 and FAU_SEL.1.1, in which **secrecyLevel** and **detailLevel** format the **event** tuple.

In terms of certification, verifying AU-2.G is correctly implemented requires that, for all events *e* in **AE**, *e* contains at least the information dictated by **event**. Thus, satisfying AU-2.G with the additional constraints means the controls related by *structures* are also satisfied.

4.4 Refines

Where *structures* targets compliance types, *refines* covers compliance relations. The primary motivation for *refines* is to make a generic constraint, e.g., the system must audit important events, more specific, e.g., the system must audit user login. Control *c1 refines* control *c2*, ($c1 \longrightarrow c2$), denotes a refinement of a declared variable *v2* in *c2*, by a declared variable *v1* in *c1*. The refinement relation requires *v1* and *v2* to be the same type. In addition, it may affect the structure of the resulting constraint specification for *c2*.

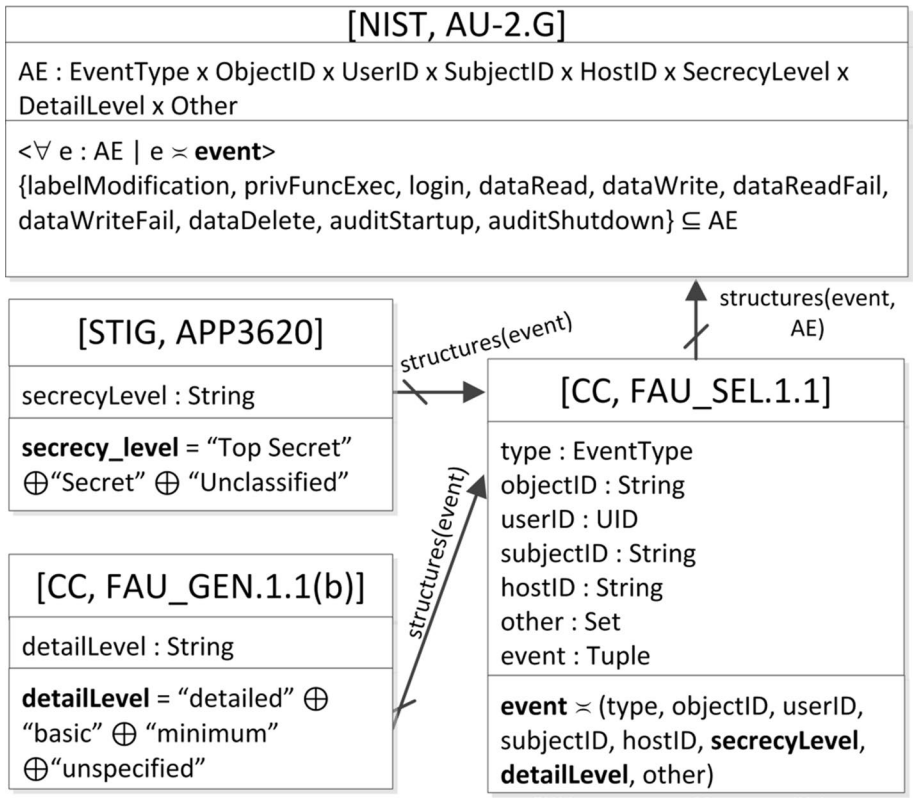

Fig. 12 Building up the structure of AuditableEvents using structures

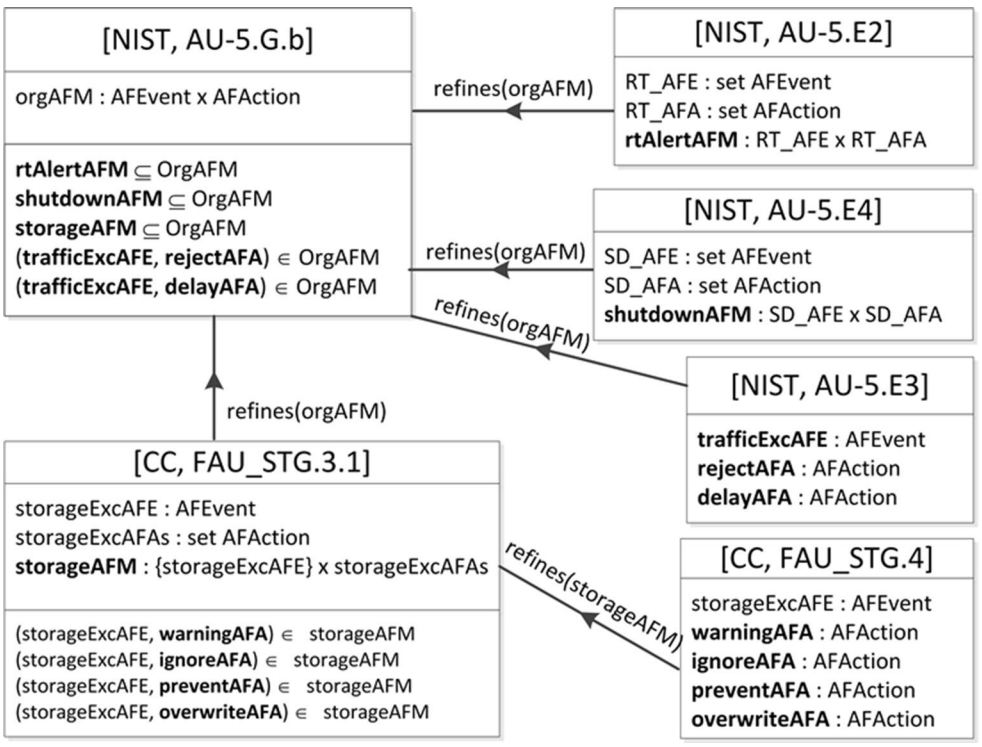

Fig. 13 Using refines for the organizational audit failure mapping control hierarchy

On the top left of Fig. 13 is AU-5.G.b, which is the dominant control in the control hierarchy of an intent class for detailing the contents of the *Audit Failure Mapping*. It comprises the organizational failure mapping (*orgAFM),* which embodies the actions (*AFActions)* that must occur in the case of certain audit failure events (*AFEvents)* as dictated

by the *refines* operator. Connected to AU-5.G.b are a number of controls that dictate particular (*AFEvent, AFAction*) pairs. Each of these pairs *refines* the *orgAFM*, establishing particular events and actions or sets of events and actions, such as real-time alerts (AU-5.E2), events that require system shutdown (AU-5.E4), traffic threshold exceeded (AU-5.E3), or storage threshold exceeded (FAU_STG.3.1 and FAU_STG.4), as part of the failure mapping.

Bold elements in the constraint specification portion denote how the refinement occurs. Bold elements in the declared variables section denote which variables influence the refinement. The label on the arrow indicates the direction of the refinement and what is being refined. Controls lower in the hierarchy refine orgAFM by adding elements. For instance, FAU_STG.3.1 and FAU_STG.4 require that certain failure actions (a warning, ignore event, prevent event, or overwrite previous events) must be available if audit record storage is exceeded. Figure 11 also shows the transitive use of refinement where FAU_STG.4 refines the event-action pairs in the audit failure mapping (*AFM*) called *storageAFM* described in FAU_STG.3.1 to include the actions of *warning*, *ignore*, *prevent*, and *overwrite*.

4.5 Forms

One of the primary research goals of this work was to produce a representative expression of each intent class in the form of a *compliance type* or a *compliance rule*. As the investigation progressed and the intent classes were defined, we found that at least one of the documents had a control that dominated the others in its class. Often it was an abstract or high-level control statement that allowed it to encompass the details of the other controls. The *forms* relation terminates the developed control hierarchy by linking this *dominant control* to the resulting compliance type or compliance rule. If a dominant control does not emerge, then the intent class should be decomposed into multiple intents of a finer granularity of expression so that a single type or rule reflects the intent. Since *forms* is only defined between a dominant control and a compliance predicate, it is not transitive.

The *forms* relation reflects the instantiated pattern of the dominant control. When the dominant control is an *imposes* pattern, it forms a *compliance type*. A compliance type defines the format, fields, and constraints on the instances of an information asset. Dominant controls that instantiate either *performs* or *protects* patterns form *compliance rules* that dictate security relations for types and functions. *Forms* is represented visually as $c \bullet\!\!-\!\!-\!\!\blacktriangleright CI$, where c is the dominant control and *CI* is the compliance item (either a rule or type), and the result of its application is that all controls in the control hierarchy below c hold whenever *CI* holds. If this perspective (i.e., complete compliance) is not desired by the information system designers, *then one or more branches can be pruned from hierarchies that define CI specific to accommodate their selected partial compliance verification process.*

Figure 14 exemplifies the two types of forms relations. On the left, the dominant control AU-5.G.a directly forms the *Audit Failure Response* compliance rule. AU-5.G.a is a dominant control based on a *performs* pattern because it fully encompasses the controls for that intent class as shown by the semantic relations of the hierarchy.

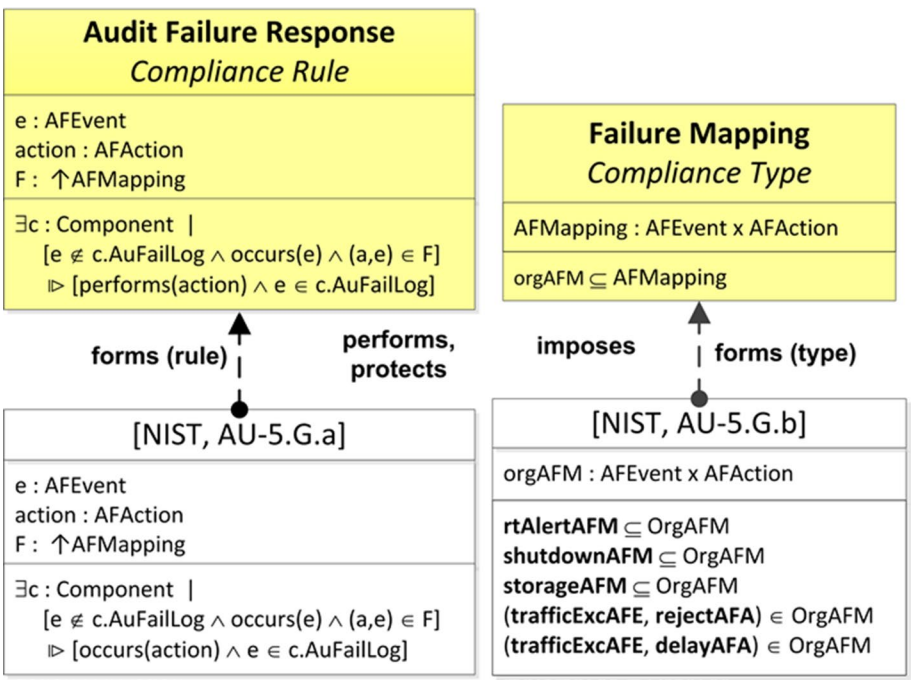

Fig. 14 Forming compliance predicates

On the right of Fig. 14 is the compliance type for *Failure Mapping*. All controls that form the failure mapping (including AU-5.G.b) are based on *imposes* patterns. It is interesting that AU-5 provides information for both a compliance type and a compliance rule thus spanning two intent classes when its details are extracted. AU-5.G.b is the dominant control of its intent class because its statement regarding the failure mapping type allows it to be structured and constrained by controls also instantiating the *imposes* pattern lower in the control hierarchy. The resulting type is then the *AFMapping*, as seen in the yellow box on the right in Fig. 14.

Table 1 provides summary guidelines for applying each type of semantic relation based on subject matter expert feedback (collected as part of the formative evaluation discussed later in Sect. 6.2).

5 Compliance interconnectivity spanning hierarchies in the audit case study

The hierarchy formed through the process of intent determination, NL patterning, and semantic relationship definition produces high-level elements, segregated into compliance rules and compliance types that provide a security verification profile with respect to the controls in the governing documents. Though our case study focuses specifically on audit, the process also produces related hierarchies across the other technical control categories including access control (AC), identification and authentication (IA), and system and communications protection (SC) in [2] and communication (class FCO), cryptographic support (class FCS), user data protection (class FDP), identification and authentication (class FIA), protection of the TSF (class FPT), resource utilization (class FRU), TOE access (class FTA), and trusted path/channels (class FTP) in [3].

Within and across technical control categories, there are relationships that should be exploited so that compliance can be easily documented by progressing through the compliance rules and types and then identifying any branches of hierarchies that may be non-compliant. This section introduces top-level semantic relations that can be used to connect compliance predicates (Compliance Rules and Types) together to denote control document interconnections. This makes the security profile specification more understandable by information system designers and certifiers and facilitates re-use by other organizations who need only apply their specific parameters to instantiate the profile. The top-level relations are applied to the running audit case study to arrive at an overall profile of audit security requirements. Each specific predicate in the overall profile is individually decomposed and discussed.

5.1 Top-level semantic relations

When extracting and formalizing the semantic hierarchies it became clear that intent classes were related at a high level. Each intent class has its own semantic hierarchy of controls that define specific compliance requirements for a system, but may also have additional interdependencies. For instance, defining auditable events in isolation is meaningless if the system does not have an audit record generation capability. Similarly, audit records cannot be protected if they do not exist. Following this rationale, it is clear that additional, high-level semantic relations are needed to express hierarchy-spanning compliance interconnectivity.

To denote these cross-hierarchy relationships our approach includes a set of top-level semantic relations that are defined between different compliance predicates, denoted as p1 and p2. Their definitions appear below and in the overall audit profile as shown in Fig. 15, discussed next.

- p1 *includedIn* p2 indicates that the constraints in the *Compliance Rule* p1 are needed for p2's verification.
- p1 *targets* p2 indicates that p1 has embedded constraints that influence or affect variables or constraints in p2.
- p1 *creates* p2 indicates that when the predicate in *Compliance Rule* p1 evaluates to true, entities described in p2 are created as a result.
- p1 *requiredBy* p2 means that the *Compliance Type* defined in p1 is required to define declared variables or elements of the constraint in the *Compliance Rule* p2.

Figure 15 exemplifies the relations and identifies the full set of compliance predicates for the entire audit technical control family across the governing documents discussed in Sect. 2.1, displaying only the control and compliance

Table 1 Semantic relations

Relation	Guidelines for Application
subsumedBy	Indicates that one control's compliance requirements are subsumed by another's requirements
usedBy	Indicates that one control compliance requirements are needed by another's
structures	Indicates that a control defines something (such as a format) regarding another control
refines	Indicates that a control clarifies or makes a requirement more explicit in another control
forms	Indicates the formation of compliance type or compliance rule from a *dominant control*

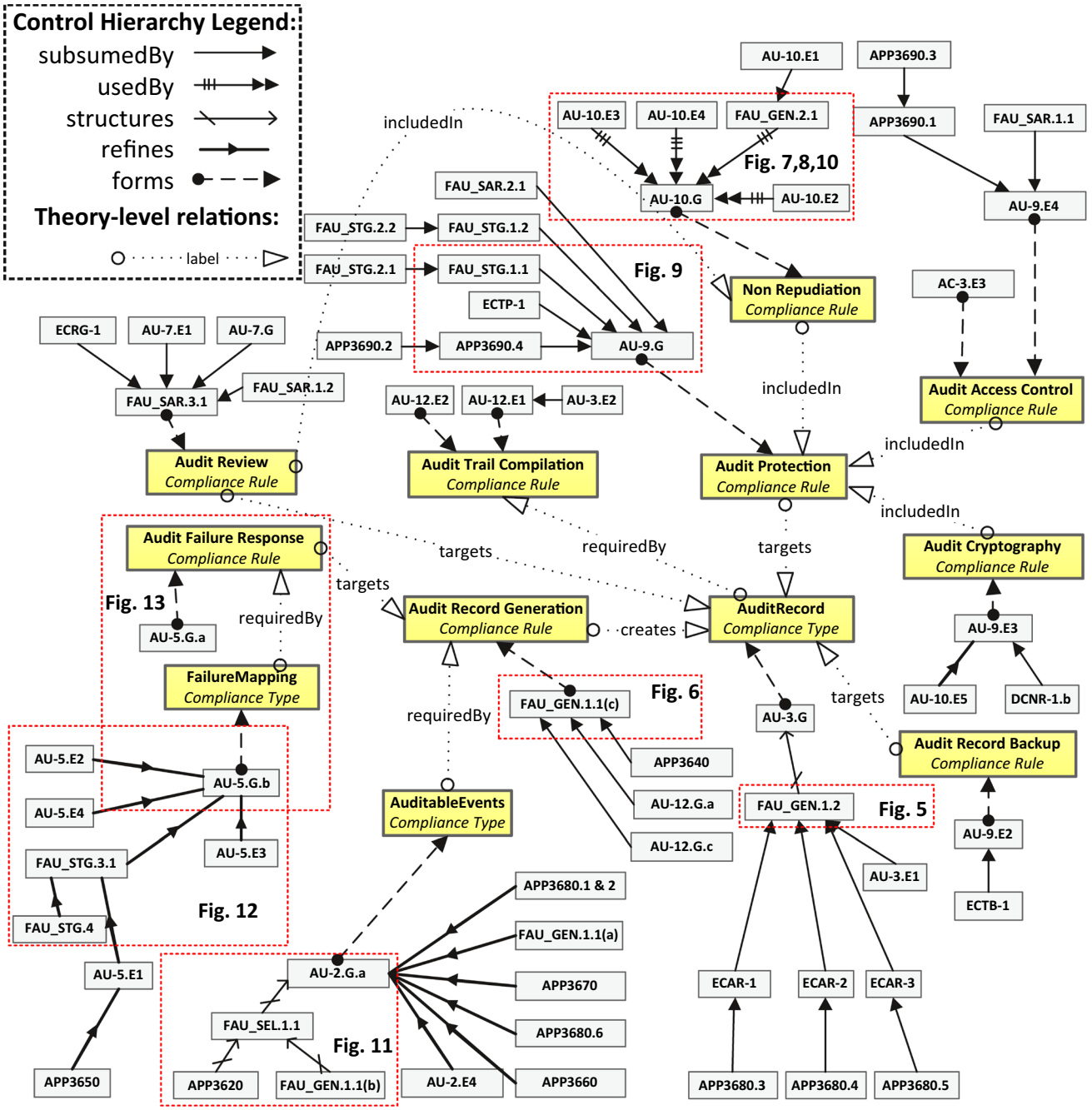

Fig. 15 Compliance predicates and their relationships for the audit technical control category and beyond. This view presents the entire compliance hierarchy relating compliance predicates (in yellow) with their associated controls (in gray). Previous figures discussed in detail are highlighted (red dotted line) to show their connectivity in the overall hierarchy. Theory level relations identify connections between disparate compliance predicates (color figure online)

predicate identifiers. Overall, there are 12 compliance predicates that describe all audit requirements and constitute the audit compliance model. Previous Figs. 4, 5, 6, 7, 8, 9, 10, 11, 12, and 13 show certain expanded sections of the compliance model and are highlighted using dotted red boxes.

Based on the constructed method of discovering, representing, and relating the predicates, Fig. 15 manifests a flow that is directly relevant to the compliance verification process. Essentially, the assessment begins by identifying *AuditableEvents*. Selected auditable events must cause audit records to be generated, via *Audit Record Generation*. These generated records take on the particular characteristics, denoted by *creates*, of the organizationally defined *AuditRecord*. A subset of these audit records, denoted by

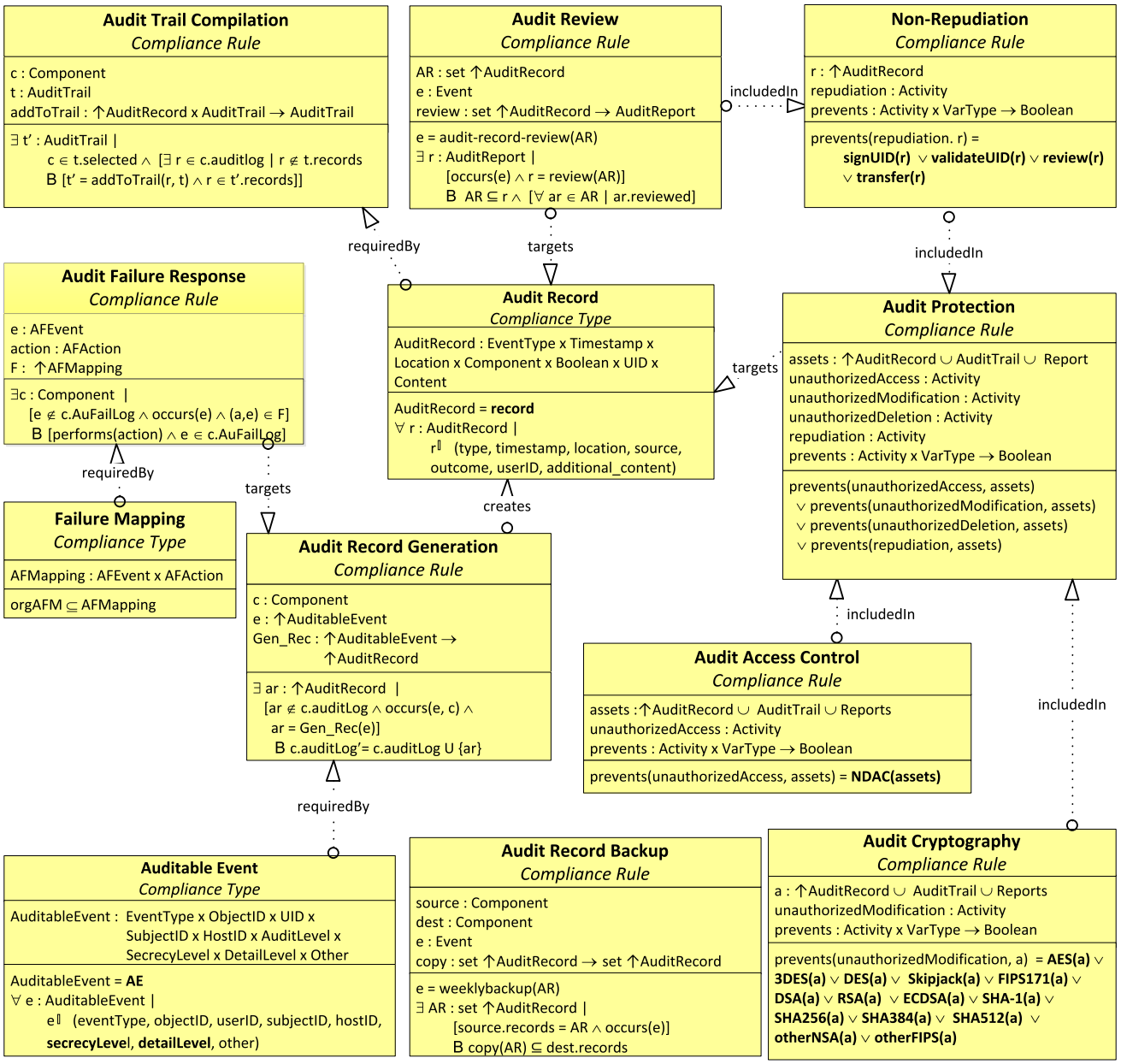

Fig. 16 Compliance types, rules, and relations, for the audit technical control category

includedIn, must be compiled into an audit trail to satisfy *Audit Trail Compilation*. Additionally, all audit records must be safeguarded (shown by *targets*) by *Audit Protection* and its three sub-parts (shown by *includedIn) Non-Repudiation*, *Audit Access Control*, and *Audit Cryptography*. Records should be reviewed using *Audit Review* and backed-up according to *Audit Record Backup*. Finally, if the audit record generation process fails, the *Audit Failure Response* relies on a *Failure Mapping* to handle audit failures in organizationally specified ways that limit asset exposure.

Figure 16 expands each compliance predicate in Fig. 15, showing all of the compliance requirements in the Audit Compliance Model. By construction, satisfaction of these criteria is sufficient to verify a system or collection of systems as compliant with audit security controls across all of the governing documents. It is important to note that organizations need not comply with all of the audit security controls, but in order to satisfy certain controls *they must* satisfy those that are indicated by the top-level and inter-hierarchy semantic relations. For instance, an organization seeking to be certified with a control that has an incoming *requiredBy* link, must necessarily be certified against the linked required control.

5.2 Auditable events

The *AuditableEvents* compliance type in Fig. 16 relies on a cross-product of the relevant event fields described within its control hierarchy. Its dominant control, AU-2.G.a from Fig. 12, restricts the type to organizationally selected auditable events as instantiated within the hierarchy. All auditable events must contain at least the information expressed in the tuple according to the governing documents. This compliance type is used as input (denoted by *requiredBy*) to Audit Record Generation, discussed in Sect. 5.4.

5.3 AuditRecord

As with *AuditableEvents,* the set of *AuditRecords*, shown in Fig. 16, is constrained to have all of the same characteristics as the organizationally defined **records** type expressed in the dominant control that forms the compliance type. For all audit records, each record must consist of at least an event type, timestamp, location, source, component, outcome, and event user ID, but can also have additional content as specified by the organization.

5.4 Audit Record Generation

The compliance rule for *Audit Record Generation*, shown in Fig. 16, is formed directly from the dominant control FAU_GEN.1.1(c), as shown in Fig. 7. Essentially, for all components, *c*, if a selected auditable event occurs, then a record is generated for that event and placed in the component's audit log, i.e., *c.auditLog*. The relation *creates* denotes that the result of *Audit Record Generation* is an entity of compliance type *Audit Record.* We assume that components must have an abstraction of the storage that houses their audit records. Otherwise, the component cannot be certified for any audit control.

5.5 Audit trail compilation

Audit Trail Compilation, connected to *AuditRecord* in Fig. 16, expresses the constraint that audit records must be compiled into a standardized audit trail. The compliance rule makes explicit that any audit records collected by a component appear in an audit trail. The temporal property guarantees that all audit trails will eventually be complete with respect to the collected audit records. The compliance rule allows for the possibility of multiple audit trails, but one must be the "system-wide" or centralized audit trail, as required by NIST [2]. Additionally, the *requiredBy* relation ensures that records in the trail conform to the type defined in *Audit Records*.

5.6 Non-repudiation, audit access control, audit cryptography, and audit protection

A set of controls exist to safeguard audit records from unintentional or malicious activities. For example, the *Audit Access Control* compliance rule specifies the non-discretionary access control policy to be applied to all audit assets. The *unauthorizedAccess* activity is an attempted unauthorized access, modification, or deletion. **NDAC(assets)**, as shown in the middle left of Fig. 16, denotes a labeled *performs* for applying a non-discretionary access control process to the assets to protect them against these activities. This item is left unexpanded, since it relies on the *Access Control* (AC) family of controls and is thus outside the scope of the audit case study. If expanded, it would have a semantic hierarchy that bridges two dominant controls AU-9.E4 (in the audit family) and AC-3.E3 (in the access control family). Access control can proactively prevent all of the activities on assets given proper system usage. However, additional provisions must be in place to prevent malicious activities that circumnavigate access control.

The compliance rule, *Non-Repudiation* (top of Fig. 16), prevents user attempts to deny having performed an activity, i.e., *repudiation* as described previously in Figs. 9 and 11.

A third compliance rule *Audit Cryptography* (lower right of Fig. 16) defines the cryptographic mechanisms that can be used for securing information assets. All of the various labeled *performs* predicates in the *prevents* expression represent various cryptographic mechanisms that may be applied. These could be expanded into the full formal predicates they represent to facilitate at-a-glance compliance, but again are left unexpanded as they cross into other families outside of audit (such as SC in the SP800-53 and FCS in the CC-Part2). Each mechanism applies to a specific cryptographic area, e.g., AES for encryption, RSA for digital signatures, and SHA-1 for hashes.

Figure 10 expresses the dominant control AU-9.G and some of the controls it subsumes. This dominant control and three related compliance rules form the *Audit Protection* compliance rule. *Audit Protection* assures three prevented activities: unauthorized access, modification, assets, or repudiation of assets. A single dominant control does not actually specify the prevention mechanisms. Instead, the three compliance rules, i.e., *Audit Access Control*, *Non-Repudiation*, and *Audit Cryptography*, are needed to fully specify the mechanisms as indicated by the *includedIn* relation in Fig. 16.

5.7 Audit review

The *Audit Review* (top of Fig. 16) compliance rule dictates the process by which audit records are reviewed. Essentially, a non-null set of records chosen for review are examined by an organizationally specified reviewer and included in an organizational asset of type *AuditReport*. An *AuditReport* is a set of reviewed records with a common purpose. In addition to being included in the report, the audit records are also marked as being reviewed, which is important as we have shown with AU-10.G for non-repudiation. This relationship is denoted by the application of *includedIn* between *Audit Review* and *Non-Repudiation*.

5.8 Audit record backup

Systems that require periodic, no less than weekly, backup must comply with the *Audit Record Backup* compliance rule. It provides a way to copy audit records from one component to another. Though these can be any two components, NIST requires them to be on different systems or media. The compliance rule in Fig. 16 for backup denotes a *source* component, a destination component (i.e., *dest*), an event *e* defined as *weeklybackup*(AR) where *e* might be run by something like a *cron* job in an actual implemented system. The temporal predicate says that if there are a set of records *AR* in the *source* component and *e* occurs, then the records are copied using a *copy* function into the destination component, i.e., *dest.records*.

5.9 Audit failure response and failure mapping

Audit Failure Response and its associated *Failure Mapping*, shown in Fig. 16, provide the last assurance requirement with respect to auditing in the governing documents. *Audit Failure Response* applies to a generic audit failure event, *e*, and its associated audit failure action, *action.* The rule says that given that if a failure event *e* occurs in some component, then the corresponding pre-defined *action* must be performed and *e* must be logged by the audit failure component *c*.

The *Failure Mapping* is the compliance type that embodies the organizationally and control defined (event, action) pairs specifically related to audit failure. It contains all of the constraints applied to *orgAFM* using the *refines* semantic relation, (as previously shown in Fig. 13). Similar to **AE** within the *Auditable Event* compliance type, *AFMapping* can be expanded to show these constraints. Some of the various (event, action) pairs include (storage exceeded, warning), (traffic exceeded, reject additional traffic), and (audit system shutdown, real-time alert) and are consistent with the refinements to *orgAFM* shown in Fig. 13.

6 Evaluation of extraction, formalization, and hierarchy creation process

A pilot study was conducted as a means of formative assessment to get feedback regarding the applicability and consistency of the compliance hierarchy formation process. After improving the extraction and modeling process, a second study summarily assessed the accuracy, efficacy, and preference of our approach. Both studies were conducted on participant pools of subject matter experts (SMEs) who had obtained, or were obtaining, CNSSI certificates. The methodology and results of each study are discussed separately below.

6.1 Pilot study methodology: formative evaluation and feedback

The first study posed three research questions for formative feedback and evaluation using SME groups:

RQ1 Is the governance patterning process understandable, consistent, and repeatable across different control families and groups of SMEs performing requirements extraction?
RQ2 Do SMEs consistently identify semantic relationships that exist between controls?
RQ3 How often are extraneous relationships identified during the semantic relationship identification process?

To answer these questions, we recruited nine SMEs from academia and industry and divided them into three panels of three. Each panel was given two collections of security controls from the governing documents (i.e., NIST sp800-53, Common Criteria, and DoDI 8500.2). The first collection consisted of 7 controls related to *identification and authentication*. The second collection varied between panels. Panel 1 was given controls related to *access control and authentication*. Panel 2 was given controls related to *user accounts*. Panel 3 was given controls related to *transmission protection.* The specific controls and their documents of origin are given in Table 2. Overall 43 unique controls across the governance documents were given to the SME panels. Each panel was given four tasks to complete for each of their control collections: (1) determine each control's pattern, (2) identify the semantic relationships among the provided controls, (3) select one or more dominant controls for the collection, and (4) form a compliance hierarchy for the control collection.

Prior to SME data collection, we completed the four tasks for each of the collections—identifying a control pattern for each included control, semantic relationships between controls, and identifying a dominant control and hierarchy

Table 2 Control collection by group and governance document in the pilot study

	(All Panels) identification and authentication	(Panel 1) access control and authentication	(Panel 2) user accounts	(Panel 2) transmission protection
NIST SP 800-53 r4	IA-2 (p. 243), IA-2(1) (p. 244), IA-3 (p. 246), IA-3(1) (p. 246)	AC-3 (p. 163), AC-3(2) (p. 163), AC-3(5) (p. 165), AC-3(7) (p. 165), AC-6 (p. 171), AC-6(2) (p. 172), AC-14.a (p. 178)	AC-2.d (p. 160), AC-2(1) (p. 161), AC-2(8) (p. 162), AC-2(2) (p. 162), AC-2(3) (p. 162), AC-2(5) (p. 162), AC-2(9) (p. 163), AC-2(10) (p. 163)	SC-8 (p. 346), SC-8(1) (p. 346), SC-12 (p. 348), SC-12(2) (p. 348), SC-17 (p. 351)
Common Criteria	FIA_AFL.1.2 (p. 89), FIA_UAU.2.1 (p. 96), FTA_MCS.1.1 (p. 164)	FDP_ACC.2.1 (p. 57), FDP_ACF.1.1 (p. 59), FIA_UID.1.1 (p. 99)	None	FCS_CKM.1.1 (p. 50), FCS_CKM.2.1 (p. 50), FPT_ITT.1.1 (p. 136), FPT_ITC.1.1 (p. 132), FPT_ITI.1.1 (p. 134)
DoDI application security STIG	None	APS0110: CAT II (p. 26)	APS0510: CAT II (p. 21), N/A[a]-I[b], II[c], III[d], IV[e] CAT II (p. 27)	APS0350: CAT I (p. 27), N/A[f]: CAT II (p. 26)

[a]These controls do not have STIG identifiers

[b]The control reads: The ASA will ensure users are not allowed to change their passwords more than once every 24 h without IAO approval

[c]The control reads: The ASA will ensure application server account passwords are changed at least once a year and anytime an ASA is reassigned

[d]The control reads: The ASA will ensure application server account passwords are a minimum of eight alphanumeric characters in length and do contain a mix of upper case letters, lower case letters, numbers, and special characters

[e]The control reads: The ASA will ensure application server account passwords do not contain personal information such as names, telephone numbers, account names, dictionary words, etc

[f]This control does not have a STIG identifier. The control reads: The ASA will ensure the application server and its hosted web applications have implemented a PKI and PK enabling solution that uses NIST-validated or NSA-approved cryptography for I&A services

for the collection. Using our task results as an *experimental control* for comparison against the SME panel results, we developed four evaluation criteria **E1**–**E4** as stated below. Henceforth, note the difference between use of the term *control* and *experimental control*; the former is a security control while the latter is a control case in the sense of experimental testing discussed in this section. Criteria **E1** and **E4** both address **RQ1**, while **E2** addresses **RQ2** and **E3** addresses **RQ3**.

> **E1** Number of control patterns selected by the panels that match the control patterns in the experimental control, where "match" is defined as the selection of the same pattern *level*, *type*, and *input*.
> **E2** Number of semantic relationships selected by the panels that match the semantic relationships in the experimental control, where "match" is defined as the selection of the same relation type, two controls, and directionality.
> **E3** What extraneous relationships exist (if any) and what relationships are missing (if any) from each panel's hierarchy as compared to the experimental control's hierarchy?
> **E4** Inter-rater reliability and internal consistency of pattern identification and hierarchy structure across panels for the shared control collection.

6.2 Pilot study results: formative evaluation and feedback

Across all SME panels, 43 unique controls were examined; of those 7 were examined by all three panels, resulting in 57 patterns (36 + 21). Of the 57 identified patterns, our experimental control identified 22 as *performs*, 30 as *imposes*, and 5 as *protects*. For evaluation criteria **E1**, we compared our expectation in the control to our observations across the panels. We found that 79% (45 of 57) of panel selected control patterns matched expectations—with SMEs identifying the same pattern type, level, and inputs as in our experimental control. All of the errors (12 of 57) involved SME panels erroneously identifying an *imposes* pattern as *performs*. This result led us to believe that there was a systemic ambiguity in the pattern selection process that made it difficult to distinguish between *performs* and *imposes*. Examining the misidentified controls further, we found that they all used active verbs, such as "produces" as in NIST SC-12(2) which states:

> The organization produces, controls, and distributes symmetric cryptographic keys using [Selection: NIST FIPS-compliant; NSA approved] key management technology and processes.

Although this control statement is imposing the use of a certain cryptographic key management process, panels interpreted the statement as a need for a particular system functionality to be performed (which was actually covered by the parent control, SC-12).

To address this issue and to make the application of *imposes* versus *performs* unambiguous, additional syntax and applicability conditions were introduced into the *imposes* and *performs* patterns (as reflected in their earlier definitions in Sects. 3.1 and 3.2). The change mainly focused on the use of the *level* as a distinguishing characteristic. Specifically, *performs* was adjusted to only apply to the *Info Sys* or *Comp* levels, whereas *imposes* specifically only applies at the *Org* level. In post-experiment discussions with the panelists, this cleared up the ambiguity.

For evaluation criteria **E2**, our experimental control expected there to be 68 semantic relations across the panels—or 10 *subsumedBy*, 31 *refines*, 10 *usedBy*, 7 *structures*, and 10 *forms*. By contrast, the SME panels collectively identified 72 semantic relations, which decomposed to 13 *subsumedBy*, 24 *refines*, 16 *usedBy*, 7 *structures*, and 12 *forms*. Comparing each of the SME identified 72 relations to our expected results, we observed 63 exact matches, i.e., the SME panel picked the same relation type, controls, and directionality as expected in the experimental control. This result translated to 92% (63 of 68) for evaluation criteria **E2**, meaning 5 relations were omitted. Looking closer at the omitted relations, 3 were *structures* and 2 were *subsumedBy*. This observation meant that the other 9 of the original 72 relations were extraneous. Five of the 9 extraneous relations were *usedBy*, suggesting a need to better clarify its semantics. Other extraneous relations included two instances of *subsumedBy*, and two *structures*. The results of analyzing the semantic relations led us to introduce additional text in each semantic relation section to clarify its formal usage. In addition, Table 1 was created to provide guidelines for the application of each relation type. Based on anecdotal post-study discussions with our SMEs, these changes reduced ambiguity and improved the accuracy and consistency of relationship identification.

Using the identification and authentication collection of 7 controls, which all panels examined, we calculated inter-rater reliability (evaluation criteria **E4**) and found it to be 71.4% across their application of the patterns and development of their hierarchies. According to the Landis and Koch-Kappa Most of the variances observed were in relation to the assignment of semantic relationships. The differences did not obstruct the creation of the hierarchies or formation of verification constructs. Overall, the strongly positive pilot study results across **E1** through **E4**, the feedback received from the participants, and the improvements made based on those discussions, suggested that the hierarchical modeling process can be consistently and reliably applied to produce the security profiles for use during verification.

6.3 Second study methodology: summative assessment and evaluation

The larger study sought to replicate the pilot study results, as well as answer three additional research questions, stated below.

RQ4 How accurately do semantic hierarchies represent the underlying control requirements and are practitioners affected by confirmation bias when examining modeled control hierarchies?
RQ5 Is the semantic compliance modeling process understandable and easy to use by certification experts?
RQ6 Is the semantic compliance modeling process preferable to less formal approaches, such as DIACAP, among practitioners?

To address these questions, the study included three tasks as described below. All tasks were completed by five security experts with compliance and certification backgrounds from industry that were familiar with the governance documents and who had obtained CNSSI certificates. Prior to the first task each subject matter expert was briefed on the extraction process and shown several examples of how the process is applied.

6.3.1 Task 1: Assessing modeling accuracy and confirmation bias

Task 1 addresses **RQ4** and focuses on identifying how accurately the model captures security requirements from the underlying controls and measures the amount of confirmation bias SMEs may have when asked to evaluate pre-existing models for accuracy. Three specific evaluation criteria are defined for Task 1.

E5 The degree to which patterns accurately represent control requirements, from 1 (low) to 10 (high).
E6 The degree to which semantic relations accurately capture connections between controls, from 1 (low) to 10 (high).
E7 What degree of confirmation bias is inherent in the assessment of **E5** and **E6**?

To assess these criteria, each SME was presented with several collections of security controls that had been pre-patterned following the semantic modeling process. They were asked to read through each of the controls (raw text from a regulatory document) and the pattern selected for the control and then rate each identified pattern on a scale from 1 (poorly captures control requirements) to 10 (accurately captures requirements). The subjects were then presented with a fully modeled semantic hierarchy of the controls and asked to rate (on a scale from 1 to 10) how well they believed the semantic relation captured connections between the controls, based on the formal definitions of each relation and their understanding of how the compliance requirements were related. Lastly, each SME was asked whether or not they agreed with the selection of the dominant control for each hierarchy. To assess confirmation bias, an inaccurate hierarchy based on a grouping of controls with incorrect relations between them, was formed and given to the subjects without their knowledge. The expectation was that SMEs would rate the accuracy of the incorrect hierarchies poorly, while rating the correct hierarchies highly. After the ratings were complete, we went over the findings with each subject and cleared any misconceptions to prepare them for the next evaluation task. A sample portion of the form used for assessment of Task 1 is provided in Fig. 17.

6.3.2 Task 2: Control pattern identification and inter-rater reliability replication

Task 2 sought to replicate the inter-rater reliability data from the pilot study, adding support for criteria **E4**. In Task 2, the SMEs were presented with all four collections of controls that the other SMEs in the pilot study had received, i.e., the 43 controls identified in Table 2. SMEs were shown the control text for each control (as it appears in the regulatory document it originated from) and then asked to select the most appropriate pattern that fit it. A sample portion of the assessment form for Task 2 is provided in Fig. 18. The expectation was that SMEs would show a similar or better inter-rater reliability as observed in the pilot study.

6.3.3 Task 3: survey questions for ease of use and preference measurement

Task 3 sought to answer **RQ5** and **RQ6** through qualitative survey feedback regarding the ease of use and preference for the semantic modeling process over other approaches. Three evaluation criteria are defined below.

E8 How intuitive are control patterns for extracting requirements in governing documents?
E9 How intuitive are the semantic relations for connecting controls together?
E10 Is the compliance modeling process preferable to informal certification approaches?

To address these criteria, Task 3 posed a number of survey questions to our SMEs. Where applicable a scale of 1 (bad) to 10 (good) was used. The following questions **Q1**–**Q7** were included in the survey. A portion of the Task 3 study form is shown in Fig. 19.

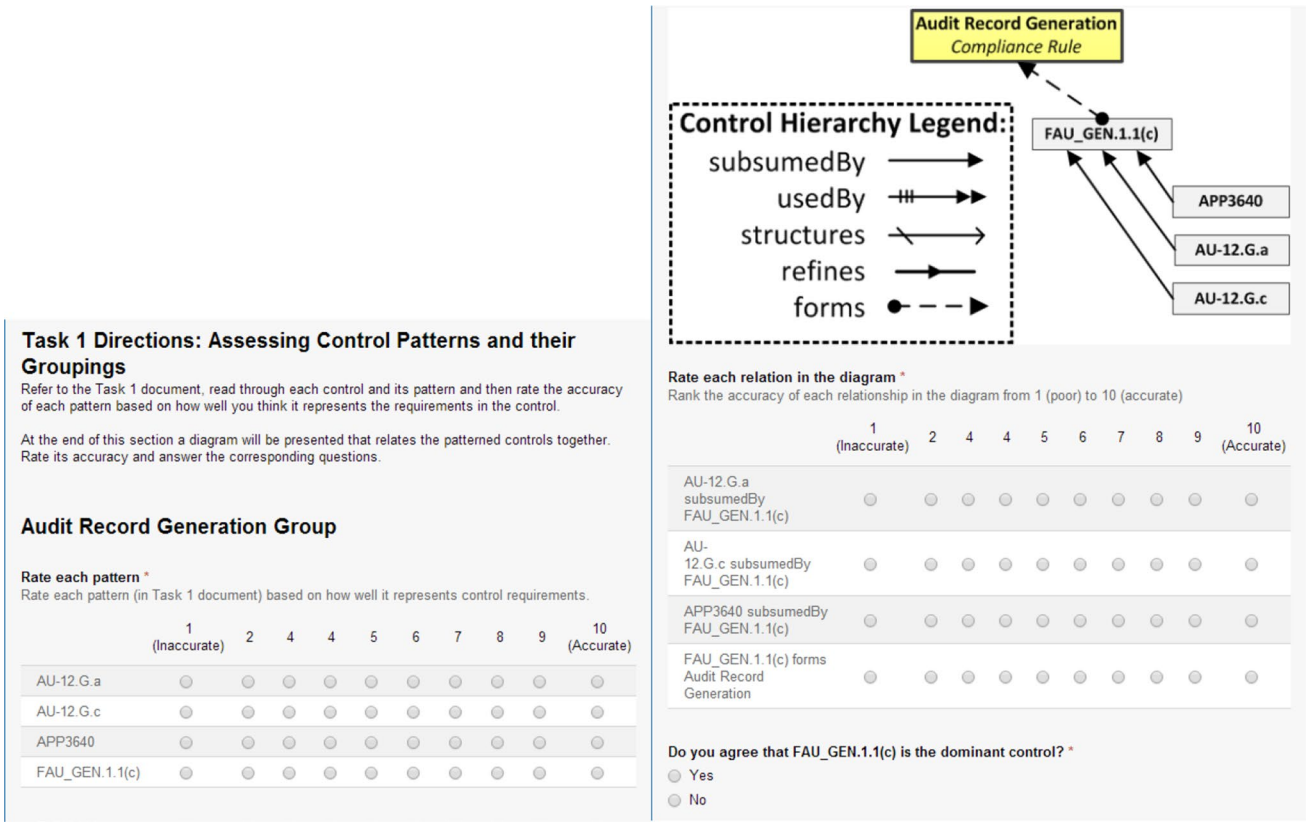

Fig. 17 Sample portion of the assessment form for Task 1

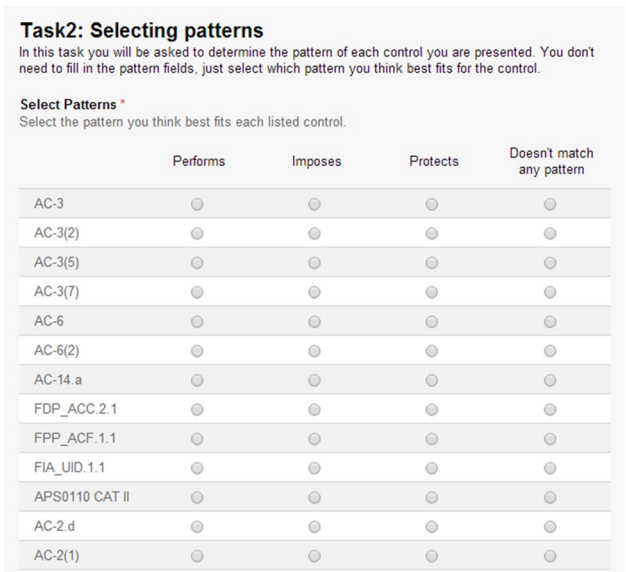

Fig. 18 Sample portion of the assessment for Task 2

Q1 How intuitive are the control patterns for extracting requirements in the governing documents?
Q2 How intuitive are the semantic relations for connecting controls together?
Q3 What about the extraction process is most difficult?
Q4 The patterns span all possible types of controls (i.e., ones not shown here).
Q5 (optional) If you disagreed, what types of patterns do you think are missing?
Q6 Extracted controls in a compliance hierarchy (the patterns and diagrams in task 1) are easier to understand than lists of un-extracted controls.
Q7 As a security engineer working to model the security compliance of a system, I would rather use this process than a checklist like the DIACAP.

6.4 Study two results: summative assessment and evaluation

The following sections examine the expert responses to the study in the context of each task and set of evaluation criteria.

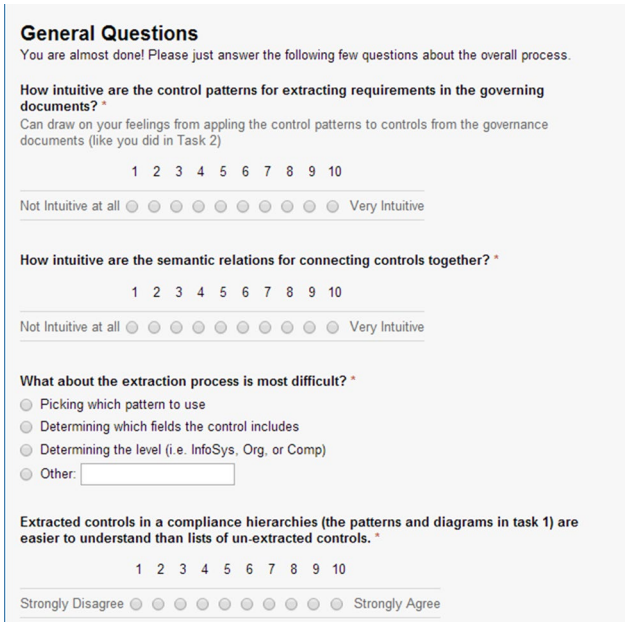

Fig. 19 Sample portion of the assessment for Task 3

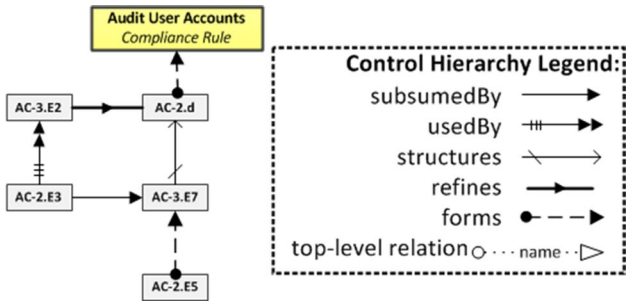

Fig. 20 Incorrect hierarchy with incorrect patterns and relationships

6.4.1 Assessing E5–E7 with task 1 results

In Task 1, the experts individually examined 61 security controls across control hierarchies in the overall compliance model, with their associated patterned statements, as expressed in "Appendix". Across all of the control hierarchies, the experts agreed with the dominant control selection 91% of the time (64 out of 70 individual assessments). Across all pattern types, the average accuracy rating of the control patterns was 9.34 (out of a possible 10) for the 305 individual pattern accuracy ratings. This suggests, for **E5**, that the model accurately represented the underlying security requirements. By pattern type, the average accuracy was 9.00 (for *protects* with 25 individual accuracy assessments), 9.34 (for *performs* with 100 assessments), and 9.39 (for *imposes* with 180 assessments) as determined by the experts.

Similarly, the experts assessed 68 semantic relations across all several hierarchies. The overall accuracy of all relation types was 9.26 across 340 individual relation accuracy ratings. By relation type, this decomposed to 9.51 (for *forms* 65 assessments), 8.90 (for *structures* 20 assessments), 9.50 (for *refines* 90 assessments), 9.10 (for *subsumedBy* 150 assessments), and 9.30 (for *usedBy* 20 assessments). This strongly suggests that experts agree with that the relationships could capture connections between security controls.

With these results in mind, our observations regarding criteria **E7**, i.e., how prevalent was confirmation bias among the experts, was surprising. We found that their assessments of the incorrect hierarchy, both in terms pattern accuracy and relationship accuracy were much higher than expected (in the 7 s for patterns and 6 s for the semantic relations). While their overall accuracy ratings were lower than the real hierarchies, the relatively high accuracy values provided show a strong confirmation bias. In other words, the security experts seemed to be inclined to believe whatever was put in front of them, if it was fully developed into a hierarchy. The specific incorrect hierarchy that duped the experts is shown in Fig. 20. This result means that the assessment results for **E5** and **E6** are less credible and require additional validation. Thankfully, the results of Task 2 and its replication of **E5**, discussed in the next section, shed further light on **E5** and **E6** by assessing the *internal consistency* of experts selecting patterns when they are not provided preformed patterns.

6.4.2 Replicating internal consistency criteria E4 with Task 2 Results

The Task 2 findings alleviated some of the concerning results of **E7**. Across the 5 experts, we found an overall internal consistency, i.e., experts picked the same control pattern, 67% of the time (with 215 individual assessments across 43 unique controls). This result decomposed to 71% internal consistency with selecting *performs* patterns (102 agreements), 75% internal consistency when selecting *protects* patterns (34 agreements), and 64% internal consistency with *imposes* patterns (76 agreements). Evaluating their selections in the same context as **E5** from the pilot study, we found that they picked the same control pattern type as the SMEs in the pilot study 64% of the time.

The raw data used to tabulate these results are presented in Table 3. All *performs* patterns are color-coded with blue, *protects* are shown in gold, *imposes* are shown in red, and *no match* is plain white. A control reference, including the name and location of the control text, is provided on the left side of Table 3. The right hand side of the table shows the highest degree of consistency across the different types. For instance, the highest consistency of the first row is 4, since 4 of the 5 experts selected *performs*.

Overall these results bolster the results of **E5** and **E6** suggesting that non-collaborating security experts select the

Table 3 Raw pattern assessment data for Task 2

Control (referrence)	Expert 1	Expert 2	Expert 3	Expert 4	Expert 5	Highest Consistency
AC-3 (sp-800-53-rev4, page 163)	Performs	Performs	Performs	Performs	Protects	4
AC-3(2) (sp-800-53-rev4, page 163)	Performs	Performs	Performs	Performs	Performs	5
AC-3(5) (sp-800-53-rev4, page 165)	Performs	Performs	Performs	Performs	Performs	5
AC-3(7) (sp-800-53-rev4, page 165)	Performs	Protects	Performs	Performs	Imposes	3
AC-6 (sp-800-53-rev4, page 171)	Imposes	Imposes	Imposes	Imposes	Performs	4
AC-6(2) (sp-800-53-rev4, page 172)	Imposes	Imposes	Imposes	Imposes	Imposes	5
AC-14.a (sp-800-53-rev4, page 178)	Imposes	Imposes	Imposes	Imposes	no match	4
FDP_ACC.2.1 (CC-Part2, page 57)	Performs	Imposes	Performs	Performs	Imposes	3
FDP_ACF.1.1 (CC-Part2, page 59)	Performs	Imposes	Performs	Performs	Imposes	3
FIA_UID.1.1 (CC-Part2, page 99)	Performs	Imposes	Performs	Performs	no match	3
APS0110: CAT II (app-stig, page 26)	Performs	Imposes	Imposes	Imposes	Protects	3
AC-2.d (sp-800-53-rev4, page 160)	Imposes	Imposes	Imposes	Imposes	Performs	4
AC-2(1) (sp-800-53-rev4, page 161)	Imposes	Imposes	Imposes	Performs	Performs	3
AC-2(8) (sp-800-53-rev4, page 162)	Performs	no match	Performs	Performs	Performs	4
AC-2(2) (sp-800-53-rev4, page 162)	Performs	Protects	Performs	Performs	Imposes	3
AC-2(3) (sp-800-53-rev4, page 162)	Performs	Protects	Performs	Performs	Imposes	3
AC-2(5) (sp-800-53-rev4, page 162)	Imposes	Protects	Imposes	Imposes	Imposes	4
AC-2(9) (sp-800-53-rev4, page 163)	Imposes	Performs	Imposes	Imposes	Imposes	4
AC-2(10) (sp-800-53-rev4, page 163)	Performs	Protects	Performs	Performs	Performs	4
APS0510: CAT II (app-stig page 21)	Performs	Performs	Imposes	Imposes	Imposes	3
N/A CAT II (app-stig page 27)	Performs	Performs	Imposes	Imposes	Imposes	3
N/A CAT II (app-stig page 27)	Performs	Performs	Imposes	Imposes	Imposes	3
N/A CAT II (app-stig page 27)	Performs	Performs	Imposes	Imposes	Imposes	3
N/A CAT II (app-stig page 27)	Performs	Performs	Imposes	Imposes	Imposes	3
IA-2 (sp-800-53-rev4, page 243)	Performs	Protects	Performs	Performs	Performs	4
IA-2(1) (sp-800-53-rev4, page 244)	Performs	Protects	Performs	Performs	Protects	3
IA-3 (sp-800-53-rev4, page 246)	Performs	Protects	Performs	Performs	Performs	4
IA-3(1) (sp-800-53-rev4, page 246)	Performs	Protects	Performs	Performs	Performs	4
FIA_AFL.1.2 (CC-Part2, page 89)	Performs	Performs	Performs	Performs	Imposes	4
FIA_UAU.2.1 (CC-Part2, page 96)	Performs	Performs	Performs	Imposes	Imposes	3
FTA_MCS.1.1 (CC-Part2, page 164)	Performs	Imposes	Performs	Imposes	Imposes	3
SC-8 (sp-800-53-rev4, page 346)	Protects	Protects	Protects	Performs	Protects	4
SC-8(1) (sp-800-53-rev4, page 346)	Protects	Protects	Performs	Performs	Protects	3
SC-12 (sp-800-53-rev4, page 348)	Protects	Protects	Performs	Performs	Imposes	2
SC-12(2) (sp-800-53-rev4, page 348)	Imposes	Protects	Imposes	Performs	Performs	2
SC-17 (sp-800-53-rev4, page 351)	Imposes	Protects	Imposes	Performs	Performs	2
FCS_CKM.1.1 (CC-Part2, page 50)	Protects	Imposes	Performs	Performs	Imposes	2
FCS_CKM.2.1 (CC-Part2, page 50)	Protects	Imposes	Performs	Performs	Imposes	2
FPT_ITT.1.1 (CC-Part2, page 136)	Protects	Protects	Protects	Performs	Protects	4
FPT_ITC.1.1 (CC-Part2, page 132)	Protects	Protects	Protects	Performs	Protects	4
FPT_ITI.1.1 (CC-Part2, page 134)	Performs	Imposes	Performs	Imposes	Imposes	3
N/A: CAT II (app-stig, page 26)	Performs	Imposes	Performs	Imposes	Protects	2
APS0350: CAT I (app-stig, page 27)	Performs	Imposes	Performs	Imposes	Performs	3

same pattern about two-thirds of the time. In a real organization, security experts would likely collaborate to ensure that everyone is on the same page and understands the information consistently. To that degree, one of the experts that took part in the study, stated in the optional comment section, that they believed iterative refinement to be absolutely essential to the compliance assessment process, in this model, or any other. This anecdote when combined with the blind two-thirds internal consistency, and the inter-rater reliability measured in the pilot study indicates that the process is relatively unambiguous and repeatable.

Another factor not discussed up to this point is the inherent ambiguity present in the security controls themselves. For instance, from Table 3, one can see that controls relating to well understood topics like access control (the AC family) were much better understood than controls relating to system

and communication protections (SC family in the NIST) and protection of the TSF (FPT in the CC). This could mean that there are inherent ambiguities in the underlying documents that make the patterning process more difficult in the same way they would make compliance assessment in another process more difficult. In other words, ambiguity in, ambiguity out.

6.4.3 Assessing E8–E10 with Task 3 results

Task 3 sought to obtain a sense of how the experts felt about the overall process in the context of other competing compliance assessment processes, like DIACAP, for instance. To this degree, the experts were asked a series of general questions as discussed in Sect. 6.3.3. For Q1, i.e., "How intuitive are the control patterns for extracting requirements in the governing documents?", the experts provided an average rating of 8.8 indicating intuitive to very intuitive. Q2, i.e., the same question, but about the semantic relations, received a slightly lower average rating of 8.2.

One of the experts, in the comments, specifically had trouble distinguishing between *structures* and *refines*. This led to additional feedback in the descriptions in Table 1 to better explain the guidelines for each. For Q3, i.e., what about the process is most difficult, the majority of experts selected the "determining which fields the control includes" option. This led us to review and adjust the language for pattern application to better express their parameters. Collectively, Q1–Q5 indicated that the patterns and relationships are intuitive for security experts, addressing **E8** and **E9** favorably.

Lastly, and perhaps most importantly, the study sought to determine if the security experts would prefer the compliance modeling process to other less formal approaches like DIACAP. Anecdotally, in the comments sections, several of the experts expressed their interest in the ability of the model to capture and represent requirements succinctly. One expert pointed to the fact that the model would only actually need to be developed once, and then, it could be re-used later by any other organization.

Looking at the more numerical assessments, there were strong results toward **E10** indicating that the model would be preferable over informal methods. Q6, asking the experts whether or not the compliance hierarchies are easier to understand than groups of unrelated controls, received an average rating of 9 across the experts indicating that they strongly agree. In the comments, two of the experts pointed to the graphical layout as providing an idea of structure that was not provided by DIACAP or other similar approaches. Similarly, Q7, which directly asked the experts if they would rather use the compliance hierarchy process over the DIACAP, received an average rating of 8.6 indicating a strong degree of agreement. Overall, the results of Tasks 1, 2, and 3 suggest that the requirement extraction process is consistent and repeatable and that the resulting compliance model semantic hierarchy is desirable to industry security analysts.

7 Discussion and conclusion

The work presented in this paper describes a process that includes reusable patterns, model templates, and semantic relations to allow new or updated controls to be patterned, formalized, and related to other controls in a new, existing, or emerging control hierarchy. The design of a compliance model provides clarity to the certification process and facilitates its application to information systems by exploiting the connectivity not only between security controls within a single document, but also across governing documents. The end result of the application of the compliance model is a set of *predicates* that solidify compliance interpretations so that they can be used directly and extended or refined by other entities that rely on the governing security documents, such as FedRamp [74] and CSA [75] for cloud computing. By establishing a compliance modeling process, our model can be expanded to cover additional security areas of concern.

This paper also followed a running audit-related case study that derived and specified all audit-related predicates in the set of examined governing documents (NIST SP800-53, etc.). The predicates specified here are thus reusable across environments, systems, and organizations utilizing those documents. Instantiating the overall compliance model given an organization and their information systems provides a structured certification baseline that can be used to formulate test cases, direct mitigation strategies, or specify the organization's desired verification constructs. Once a document is extracted, and the resulting semantic hierarchy-based compliance model is verified, it can be re-used until the underlying security control texts are changed or extended—at which point the model can be updated. This approach provides a stable, reusable, graphic, and formalizable certification baseline that can be used in conjunction with formal or informal accreditation processes for improved certification against control standards.

As NIST has now separated out the original privacy controls from the general families and created a new, dedicated segment of the SP800-53r4 to only privacy related controls, application of this research to those privacy controls is part of future effort. Another future effort is to examine the impact that constructed overlays have in clarifying certification needs by applying the controls within a specific domain. It is possible that the overlays that form the control sets could be "overlaid" onto the set of predicates to clarify how predicates can be directly instantiated.

Acknowledgements This material is based on research sponsored in part by the Air Force Office of Scientific Research (AFOSR), under Agreement No. FA-9550-09-1-0409.

Appendix A: Study: control patterns for all audit-related controls in the NIST SP 800-53, DoDI8500.2, ISO 15408-2, and related documents

Group: auditable events

AU-2.G.a: The organization determines, based on a risk assessment and mission/business needs, that the information system must be capable of auditing the following events: [*Assignment: organization-defined list of auditable events*].

Pattern:

Org *imposes* a set of auditable events *on* information systems

FAU_SEL.1.1: The TSF shall be able to select the set of audited events from the set of all auditable events based on the following attributes: a) [selection: *object identity, user identity, subject identity, host identity, event type*] b) [assignment: *list of additional attributes that audit selectivity is based upon*].

Pattern:

Org *imposes* audit parameters (object_id, user_id, subject_id, host_id, event_type, and additional_content) *on* auditable events

FAU_GEN.1.1(b): The TSF shall be able to generate an audit record of all auditable events for the [selection, choose one of: *minimum, basic, detailed, not specified*] level of audit.

Pattern:

Org *imposes* audit levels as parameters (minimum, basic, detailed, not specified) *on* auditable events

APP3620: The Designer will ensure the application does not disclose unnecessary information to users.

Pattern:

Org *imposes* a secrecy level *on* system information

AU-2.E4: The organization includes execution of privileged functions in the list of events to be audited by the information system.

Pattern:

Org *imposes* execution of privileged functions as a selected auditable event

APP3660: The Designer will ensure the application has a capability to notify the user on login of date and time of the user's last unsuccessful logon, IP address of the user's last unsuccessful logon, date and time of the user's last successful logon, IP address of the user's last successful logon, and number of unsuccessful logon attempts since the last successful logon.

Pattern:

Org *imposes* use of login function as a selected auditable event with parameters (timestamp and IP address of last unsuccessful logon, timestamp and IP of last successful logon, number of unsuccessful logon attempts since last successful logon)

APP3680.1: The Designer will ensure the application design includes audits on all access to need-to-know information.

Pattern:

Org *imposes* access control reads and writes as selected auditable events

APP3680.2: The Designer will ensure the application logs all failed access attempts to need-to-know information.

Pattern:

Org *imposes* access control read-fails and write-fails as selected auditable events

APP3670: The Designer will ensure the application has a capability to display the user's time and date of the last change in data content.

Pattern:

Org *imposes* data modification or deletion as selected auditable events

APP3680.6: The Designer will ensure the application creates an audit trail for addition, deletion, or change of the confidentiality or integrity labels as designated by the information owner.

Pattern:

Org *imposes* confidentiality and integrity label modification as selected auditable events

FAU_GEN.1.1(a): The TSF shall be able to generate an audit record of start-up and shutdown of the audit functions.

Pattern:

Org *imposes* data modification or deletion as selected auditable events

Group: audit record

AU-3.G: The information system produces audit records that contain sufficient information to, at a minimum, establish what type of event occurred, when (date and time) the event occurred, where the event occurred, the source of the event, the outcome (success or failure) of the event, and the identity of any user/subject associated with the event.

Pattern:

Org *imposes* content parameters (type, timestamp, location, source, outcome, UID) on audit records

AU-3.E1: The information system includes [*Assignment: organization-defined additional, more detailed information*] in the audit records for audit events identified by type, location, or subject.

Pattern:

Org *imposes* additional content *on* audit records

FAU_GEN.1.2: The TSF shall record within each audit record at least the following information: a) Date and time of the event, type of event, subject identity (if applicable), and the outcome (success or failure) of the event; and b) For each audit event type, based on the auditable event definitions of the functional components included in the PP/ST, [assignment: *other audit relevant information*].

Pattern:

Org *imposes* content parameters (timestamp, location, source, outcome, UID) on audit records

ECAR-1: Audit records for MAC-1 include: user ID, date and time of the event, and the type of event.

Pattern:

Org *imposes* content parameters (type, timestamp, UID) on audit records

ECAR-2: Audit records for MAC-2 include: user ID, date and time of the event, type of event, success or failure of event, [and information needed to record defined auditable events—i.e. source].

Pattern:

Org *imposes* content parameters (type, timestamp, outcome, UID, source) on audit records

ECAR-3: Audit records for MAC-3 events include: user ID, date and time of the event, type of event, success or failure of event [and information needed to record defined auditable events—i.e. source].

Pattern:

Org *imposes* content parameters (type, timestamp, outcome, UID, source) on audit records

APP3680.3: The Designer will ensure the application's publicly releasable data audit records include: user ID, date and time of the event, and the type of event.

Pattern:

Org *imposes* content parameters (type, timestamp, UID) on audit records

APP3680.4: The Designer will ensure the application's sensitive data audit records include: user ID, date and time of the event, type of event, and the success or failure of event [and information needed to record defined auditable events—i.e. source].

Pattern:

Org *imposes* content parameters (type, timestamp, outcome, UID, source) on audit records

APP3680.5: The Designer will ensure the application's classified data audit records include: user ID, date and time of the event, type of event, success or failure of event [and information needed to record defined auditable events—i.e. source].

Pattern:

Org *imposes* content parameters (type, timestamp, outcome, UID, source) on audit records

Group: audit record generation

AU-12.G.a: The information system provides audit record generation capability for the list of auditable events defined in AU-2 at [*Assignment: organization-defined information system components*].

Pattern:

InfoSys *performs* audit record generation *given* an auditable event occurs and parameters (set of declared auditable events and components (imposed by Org)) *that results in* audit record creation

AU-12.G.c: The information system generates audit records for the list of audited events defined in AU-2 with the content as defined in AU-3.

Pattern:

InfoSys *performs* audit record generation *given* an auditable event occurs and parameters (set of declared auditable events and components (imposed by Org)) *that results in* audit record creation

APP3640: The Designer will ensure the application supports the creation of transaction logs for access and changes to the data.

Pattern:

InfoSys *performs* audit record generation *given* an auditable event occurs and parameters (set of declared

auditable events and components (imposed by Org)) *that results in* audit record creation

FAU_GEN.1.1(c): The TSF shall be able to generate an audit record of the following auditable events [assignment: *other specifically defined auditable events*].

Pattern:

InfoSys *performs* audit record generation *given* an auditable event occurs and parameters (set of declared auditable events and components (imposed by Org)) *that results in* audit record creation

Group: audit trail compilation

AU-3.E2: The organization centrally manages the content of audit records generated by [*Assignment: organization-defined information system components*].

Pattern:

Org *imposes* centrally managed content *on* select components

AU-12.E1: The information system compiles audit records from [*Assignment: organization-defined information system components*] into a system-wide (logical or physical) audit trail that is time correlated to within [*Assignment: organization-defined level of tolerance for relationship between time stamps of individual records in the audit trail*].

Pattern:

InfoSys *performs* audit record compilation *given* component audit record generation and parameter (list of components selected to be part of the trail) *that results in* production of a time correlated, system-wide audit trail

AU-12.E2: The information system produces a system-wide (logical or physical) audit trail composed of audit records in a standardized format.

Pattern:

InfoSys *performs* audit record conversion *given* unstandardized audit records and parameters (list of components selected to be part of the audit trail, standard format) *that results in* audit records being in a standard format

Group: non-repudiation

AU-10.G: The information system protects against an individual falsely denying having performed a particular action.

Pertinent Supplemental Guidance Non-repudiation services are obtained by employing various techniques or mechanisms (e.g., digital signatures, digital message receipts).

Pattern:

InfoSys *protects* user action logs *using* digital signature mechanisms (imposed by org) *to prevent* repudiation

AU-10.E1: The information system associates the identity of the information producer with the information.

Pattern:

InfoSys *performs* identity (UID) binding *given* audit record generation and parameters (audit record, user, binding mechanism) *that results in* a signed UID in the audit record

AU-10.E2: The information system validates the binding of the information producer's identity to the information.

Pertinent Supplemental Guidance This control enhancement is intended to mitigate the risk that information is modified between production and review. The validation of bindings can be achieved, for example, by the use of cryptographic checksums.

Pattern:

InfoSys *performs* binding validation *given* audit record review and parameter (validation mechanism) *that results in* UID validation

AU-10.E3: The information system maintains reviewer/releaser identity and credentials within the established chain of custody for all information reviewed or released.

Pertinent Supplemental Guidance If the reviewer is a human or if the review function is automated but separate from the release/transfer function, the information system associates the identity of the reviewer of the information to be released with the information and the information label.

Pattern:

InfoSys *performs* reviewer identity binding *given* audit record review and parameters (audit record, user, binding mechanism) *that results in* signed reviewer UID in the audit record

AU-10.E4: The information system validates the binding of the reviewer's identity to the information at the transfer/release point prior to release/transfer from one security domain to another security domain.

Pattern:

InfoSys *performs* binding validation *given* audit record transfer and parameter (audit record, validation mechanism) *that results in* UID validation

FAU_GEN.2.1 For audit events resulting from actions of identified users, the TSF shall be able to associate each

auditable event with the identity of the user that caused the event.

Pattern:

InfoSys *performs* identity (UID) binding *given* audit record generation and parameters (audit record, user, binding mechanism) *that results in* a signed UID in the audit record

Group: audit failure system

AU-5.G.a: The information system alerts designated organizational officials in the event of an audit processing failure.

Pattern:

Org *performs* a set of audit failure actions related to audit failure events

AU-5.G.b: The information system takes the following additional actions (in the event of an audit processing failure): [*Assignment: organization-defined actions to be taken (e.g., shut down information system, overwrite oldest audit records, stop generating audit records)*].

Pattern:

Org *imposes* additional actions related to audit failure events

AU-5.E1: The information system provides a warning when allocated audit record storage volume reaches [*Assignment*: organi*zation-defined percentage*] of maximum audit record storage capacity.

Pattern:

Org *imposes* the percentage of audit record storage capacity which, when reached, causes a "warning action"

AU-5.E2: The information system provides a real-time alert when the following audit failure events occur: [*Assignment: organization-defined audit failure events requiring real-time alerts*].

Pattern:

Org *imposes* select failure events which cause real-time alert actions

AU-5.E3: The information system enforces configurable traffic volume thresholds representing auditing capacity for network traffic and [*Selection: rejects or delays*] network traffic above those thresholds.

Pattern:

Org *imposes* configurable traffic volume thresholds for auditing capacity of network traffic which, when reached, causes a reject or delay action

AU-5.E4: The information system invokes a system shutdown in the event of an audit failure, unless an alternative audit capability exists.

Pattern:

Org *imposes* a system shutdown action for audit failure events without alternative audit capability

FAU_STG.3.1: The TSF shall [assignment: *actions to be taken in case of possible audit storage failure*] if the audit trail exceeds [assignment: *pre-defined limit*].

Pattern:

Org *imposes* selected actions to be applied when the audit trail exceeds a pre-defined limit

FAU_STG.4.1: The TSF shall [selection, choose one of: *"ignore audited events", "prevent audited events, except those taken by the authorised user with special rights", "overwrite the oldest stored audit records"*] and [assignment: *other actions to be taken in case of audit storage failure*] if the audit trail is full.

Pattern:

Org *imposes* ignore, prevent and overwrite actions to be applied when the audit trail is full

APP3650: The Designer will ensure the application has a capability to notify an administrator when audit logs are nearing capacity as specified in the system documentation.

Pattern:

Org *imposes* the percentage of audit record storage capacity which, when reached, causes a "warning action"

Group: audit protection

AU-9.G: The information system protects audit information and audit tools from unauthorized access, modification, and deletion.

Pertinent Supplemental Guidance Audit information includes all information (e.g., audit records, audit settings, and audit reports) needed to successfully audit information system activity.

Pattern:

InfoSys *protects* audit records, the audit trail, auditable event definitions, and audit review tools *using* {} *to prevent* unauthorized access, modification or deletion

APP3690.4: The IAO will ensure the audit trail is protected against modification or deletion except by application administrators and auditors.

Pattern:

InfoSys *protects* the audit trail *using* {} *to prevent* modification or deletion

APP3690.2: The Designer will ensure the audit trail is protected against modification or deletion except by application administrators and auditors.

Pattern:

InfoSys *protects* the audit trail *using* {} *to prevent* modification or deletion

ECTP-1: The contents of audit trails are protected against unauthorized access, modification or deletion.

Pattern:

InfoSys *protects* the audit trail *using* {} *to prevent* unauthorized access, modification, or deletion

FAU_SAR.2.1: The TSF shall prohibit all users read access to the audit records, except those users that have been granted explicit read-access.

Pattern:

InfoSys *protects* audit records *using* {} *to prevent* unauthorized access

FAU_STG.1.1: The TSF shall protect the stored audit records in the audit trail from unauthorized deletion.

Pattern:

InfoSys *protects* the audit trail *using* {} *to prevent* unauthorized deletion

FAU_STG.1.2: The TSF shall be able to [selection, choose one of: *prevent, detect*] unauthorized modifications to the stored audit records in the audit trail.

Pattern:

InfoSys *protects* the audit trail *using* {} *to prevent* unauthorized modification

FAU_STG.2.1 The TSF shall protect the stored audit records in the audit trail from unauthorized deletion.

Pattern:

InfoSys *protects* the audit trail *using* {} *to prevent* unauthorized deletion

FAU_STG.2.2 The TSF shall be able to [selection, choose one of: *prevent, detect*] unauthorized modifications to the stored audit records in the audit trail.

Pattern:

InfoSys *protects* the audit trail *using* {} *to prevent* unauthorized modification

Group: audit access control

AC-3.E3: The information system enforces [*Assignment: organization-defined nondiscretionary access control policies*] over [*Assignment: organization-defined set of users and resources*] where the policy rule set for each policy specifies: (a) Access control information (i.e., attributes) employed by the policy rule set (e.g., position, nationality, age, project, time of day); and (b) Required relationships among the access control information to permit access.

Pattern:

InfoSys *performs* access control enforcement *given* user access to assets or functions and org-defined access control policy parameters (position, nationality, age, project, time of day, relationships between information) *that results in* acceptance or rejectance of user access

AU-9.E4: The organization: a) authorizes access to management of audit functionality to only a limited subset of privileged users; and b) protects the audit records of non-local accesses to privileged accounts and the execution of privileged functions.

Pattern:

Org *imposes* an access control list that specifies users authorized to access audit parameters (records, trail, tools).

FAU_SAR.1.1: The TSF shall provide [assignment: *authorised users*] with the capability to read [assignment: *list of audit information*] from the audit records.

Pattern:

Org *imposes* an access control list that specifies users authorized to access audit parameters (records).

APP3690.1: The Designer will ensure the audit trail is readable only by the application administrators and auditors.

Pattern:

Org *imposes* an access control list that specifies users authorized to access audit parameters (trail).

APP3690.3: The IAO will ensure the audit trail is readable only by application administrators and auditors.

Pattern:

Org *imposes* an access control list that specifies users authorized to access audit parameters (trail).

Group: audit cryptography

AU-9.E3: The information system uses cryptographic mechanisms to protect the integrity of audit information and audit tools.

Pertinent Supplemental Guidance An example of a cryptographic mechanism for the protection of integrity is the computation and application of a cryptographic-signed hash using asymmetric cryptography, protecting the confidentiality of the key used to generate the hash, and using the public key to verify the hash information.

Pattern:

InfoSys *performs* encryption *given* audit information(records, trail) modification and defined cryptographic mechanisms(imposed by org) *that results in* encrypted audit information

DCNR-1.b: NIST FIPS 140-2 validated cryptography (e.g., DoD PKI class 3 or 4 token) is used to implement encryption (e.g., AES, 3DES, DES, Skipjack), key exchange (e.g., FIPS 171), digital signature (e.g., DSA, RSA, ECDSA), and hash (e.g., SHA-1, SHA-256, SHA-384, SHA-512). Newer standards should be applied as they become available.

Pattern:

Org *imposes* the encryption mechanisms for the systems as: encryption: selection(AES, 3DES, DES, Skipjack); key exchange: selection(FIPS 171); digital signatures: selection(DSA, RSA, ECDSA); hash: selection(SHA-1, SHA-256, SHA-384, SHA-512)

AU-10.E5 The organization employs [*Selection: FIPS-validated; NSA-approved*] cryptography to implement digital signatures.

Pattern:

Org *imposes* the encryption mechanisms for digital signatures: selection(FIPS-validated, NSA-approved);

Group: audit review

FAU_SAR.3.1: The TSF shall provide the ability to apply [assignment: *methods of selection and/or ordering*] of audit data based on [assignment: *criteria with logical relations*].

Pattern:

InfoSys *performs* audit reduction and report generation *given* parameter (audit trail, selection/ordering method, selection criteria) *that results in* the creation of an audit report

AU-7.G: The information system provides an audit reduction and report generation capability

Pattern:

InfoSys *performs* audit reduction and report generation *given* parameter (audit trail) *that results in* the production of an audit report

AU-7.E1: The information system provides the capability to automatically process audit records for events of interest based on selectable event criteria.

Pattern:

InfoSys *performs* audit reduction and report generation *given* parameter (audit trail, selection criteria) *that results in* the production of an audit report

ECRG-1: Tools are available for the review of audit records and for report generation from audit records.

Pattern:

InfoSys *performs* audit report generation *given* parameter (audit trail) *that results in* the production of an audit report

FAU_SAR.1.2: The TSF shall provide the audit records in a manner suitable for the user to interpret the information.

Pattern:

Org *imposes* human readable qualities on generated audit reports

Group: audit record backup

AU-9.E2: The information system backs up audit records [*Assignment: organization-defined frequency*] onto a different system or media than the system being audited.

Pattern:

InfoSys *performs* audit record backup *given* the backup frequency (imposed by the Org) and parameter (audit trail) *that results in* the creation of an audit records backup on a different system or media

ECTB-1: The audit records are backed up not less than weekly onto a different system or media than the system being audited.

Pattern:

InfoSys *performs* audit record backup *given* the backup frequency (no less than weekly) and parameter (audit trail) *that results in* the creation of an audit records backup on a different system or media

Group: audit user accounts (incorrect grouping, methodological control)

AC-2.d: The organization specifies authorized users of the information system, group and role membership, and access authorizations (i.e., privileges) and other attributes (as required) for each account.

Pattern:

Org *imposes* organizational and individual access control parameters on user accounts

AC-3.E2: The information system enforces dual authorization for *[Assignment: organization-defined privileged commands and/or other organization-defined actions]*.

Pattern:

InfoSys *performs* dual authorization *given* privileged functions and actions *that results in* user login.

AC-3.E7: The information system enforces a role-based access control policy over defined subjects and objects and controls access based upon *[Assignment: organization-defined roles and users authorized to assume such roles]*.

Pattern:

Org *imposes* role-based access control policy on subjects

AC-2.E3: The information system automatically disables inactive accounts after *[Assignment: organization-defined time period]*.

Pattern: (wrong should be performs)

Org *imposes* inactive account disabling

AC-2.E5: The organization requires that users log out when [*Assignment: organization-defined time-period of expected inactivity or description of when to log out*].

Pattern: (wrong should be imposes)

InfoSys *performs* user log out *given* inactive time period *that results in* log out.

References

1. DoD (2007) Instruction 8510.01: department of defense information assurance certification and accreditation process (DIACAP)
2. NIST (2013) Special publication 800-53 recommended security controls for federal information systems rev. 4., http://csrc.nist.gov/publications/nistpubs/800-53-Rev3/sp800-53-rev3-final.pdf. Accessed Oct 2015
3. (2009) Common criteria for information technology security evaluation version 3.1 (Part 2: security functional requirements). http://www.commoncriteriaportal.org/cc/. Accessed Oct 2015
4. DoD (2003) Instruction 8500.2, information assurance implementation
5. DISA (2011) Application security and development STIG version 3 release 4. http://iase.disa.mil/stigs/app_security/app_sec/app_sec.html. Accessed Oct 2015
6. Hassan W, Logrippo L (2009) A governance requirements extraction model for legal compliance validation. In: Second international workshop on requirements engineering and law
7. Gamble M, Gamble R, Hale M (2011) Security policy foundations in context UNITY. In: 7th International workshop on software engineering for secure systems
8. Taguchi K, Yoshioka N, Tobita T, Kaneko H (2010) Aligning security requirements and security assurance using the common criteria. Presented at the fourth international conference on secure software integration and reliability improvement
9. Li N, Wang Q (2006) Beyond separation of duty: an algebra for specifying high-level security policies. In: Proceedings of the 13th ACM conference on computer and communications security
10. (2003) Summary of the HIPAA privacy rule. http://www.hhs.gov/ocr/privacy/hipaa/understanding/summary/privacysummary.pdf. Accessed Oct 2015
11. Best B, Jürjens J, Nuseibeh B (2007) Model-based security engineering of distributed information systems using UMLsec. In: 29th International conference on software engineering
12. Adir A, Asaf S, Fournier L, Jaeger I, Peled O (2007) A framework for the validation of processor architecture compliance. In: Proceedings of the 44th annual design automation conference
13. NIST (2010) Special publication 800-37, guide for applying the risk management framework to federal information systems a security life cycle approach
14. Hale M, Gamble R (2012) Risk propagation of security SLAs in the cloud In: Proceeding of the workshop on management and security technologies for cloud computing 2012, IEEE GLOBECOM
15. MITRE. The common weakness enumeration (CWE) initiative. MITRE Corporation. http://cwe.mitre.org/. Accessed Oct 2015
16. MITRE. Common vulnerabilities and exposures (CVE) initiative/ MITRE Corporation. http://cve.mitre.org/. Accessed Oct 2015
17. Minkiewicz A (2011) Cloud Nine, are we there yet? J Softw Technol 14(4):4–8
18. CSA (2012) Cloud controls matrix. https://cloudsecurityalliance.org/wp-content/themes/csa/download-box-ccm-v1-3.php. Accessed Oct 2015
19. Breaux T, Anton A (2005) Deriving semantic models from privacy policies. In: Proceedings of the sixth IEEE international workshop on policies for distributed systems and networks
20. DoD (1999) Directive 5200.2: personnel security program
21. DoD (1997) Instruction 5200.40: DoD information technology security certification and accreditation process (DITSCAP)
22. DoD (2002) Directive 8500.1: information assurance
23. OMB (1996) Circular no. A-130: memorandum for heads of executive departments and establishments—management of federal information resources
24. (2009) Common criteria for information technology security evaluation version 3.1 (Part 1: introduction and general model). http://www.commoncriteriaportal.org/cc/. Accessed Oct 2015
25. (2009) Common criteria for information technology security evaluation version 3.1 (Part 3: security assurance requirements). http://www.commoncriteriaportal.org/cc/. Accessed Oct 2015
26. Haley C, Laney R, Moffett J, Nuseibeh B (2008) Security requirements engineering: a framework for representation and analysis. Trans Softw Eng (IEEE) 34(1):133–153
27. Haley C, Moffett J, Laney R, Nuseibeh B (2006) A framework for security requirements engineering. In: Proceedings of the 2006 international workshop on Software engineering for secure systems
28. Redl C, Breskovic I, Brandic I, Dustdar S (2012) Automatic SLA matching and provider selection in grid and cloud computing markets. In: Proceedings of the 13th international conference on grid computing
29. Modica G, Petralia G, Tomarchio O (2012) A business ontology to enable semantic matchmaking in open cloud markets. In: Eighth international conference on semantics, knowledge and grids
30. Belhajjame K, Embury S, Paton N (2013) Verification of semantic web service annotations using ontology-based partitioning. In: IEEE transactions on services computing (preprint)
31. Zhu W (2012) Semantic mediation bus (TM): an ontology-based runtime infrastructure for service interoperability. In: Proceedings of 16th IEEE international enterprise distributed object computing conference workshops
32. Dobson G, Sanchez-Macian A (2006) Towards unified QoS/SLA ontologies. In: Proceedings of the IEEE services computing workshops
33. Khoury P, Mokhtari A, Coquery E, Hacid M-S (2008) An ontological interface for software developers to select security patterns.

In: Proceedings of the 19th international conference on database and expert systems application
34. W3C (2004) OWL web ontology language guide. http://www.w3.org/TR/owl-guide/. Accessed Oct 2015
35. Wongthongtham P, Chang E, Dillon T, Sommerville I (2009) Development of a software engineering ontology for multisite software development. IEEE Trans Knowl Data Eng 21(8):1205–1217
36. Mace J, Parkin S, Moorsel AV (2010) A collaborative ontology development tool for information security managers. In: Proceedings of the 4th symposium on computer human interaction for the management of information technology
37. Evesti A, Ovaska E, Savola R (2009) From security modelling to run-time security monitoring. In: Proceeedings of European workshop on security in model driven architecture (SECMDA)
38. Lee S, Gandhi R, Muthurajan D, Yavagal D, Ahn G (2006) Building problem domain ontology from security requirements in regulatory documents. In: Proceedings of the international workshop on software engineering for secure systems
39. Tsoumas B, Gritzalis D (2006) Towards an ontology-based security management. In: Proceedings of the 20th international conference on advanced information networking and applications, vol 01
40. Lee S, Gandhi R, Wagle S (2007) Towards a requirements-driven workbench for supporting software certification and accreditation. In: Proceedings of the third international workshop on software engineering for secure systems
41. Weber-Jahnke J, Onabajo A (2009) Mining and analysing security goal models in health information systems. In: Proceedings of the ICSE workshop on software engineering in health care
42. Mathews AW, Yadron D (2015) Health insurer anthem hit by hackers. Wall Str J
43. Daramola O, Sindre G, Stalhane T (2012) Pattern-based security requirements specification using ontologies and boilerplates. In: 2012 Second IEEE international workshop on requirements patterns (RePa), pp 54–59
44. Sharma V, Sarkar S, Verma K, Panayappan A, Kass A (2009) Extracting high-level functional design from software requirements. In: 16th Asia-Pacific software engineering conference
45. Ambriola V, Gervasi V (1997) Processing natural language requirements. In: Proceedings of the 12th international conference on automated software engineering
46. Bernsmed K, Jaatun M, Meland P, Undheim A (2012) Thunder in the clouds: security challenges and solutions for federated clouds. In: 4th International IEEE conference on cloud computing technology and science
47. Bleikertz S, Groß T, Mödersheim S (2011) Automated verification of virtualized infrastructures. In: Proceedings of the 3rd ACM workshop on cloud computing security workshop
48. She W, Yen I, Thuraisingham B, Huang S (2011) Rule-based runtime information flow control in service cloud. In: IEEE international conference on web services
49. Singaravelu L, Pu C (2007) Fine-grain, end-to-end security for web service compositions In: IEEE international conference on services computing
50. Roman G-C, Julien C, Payton J (2007) Modeling adaptive behaviors in context UNITY. Theor Comput Sci 376(3):185–204
51. deNicola R, Ferrari G, Pugliese R (1998) KLAIM: a kernel language for agents interaction and mobility. IEEE Trans Softw Eng 24(5):315–330
52. Bravetti M, Busi N, Gorrieri R, Lucchi R, Zavattaro G (2004) Security issues in the tuple-space coordination model. In: Proceedings of workshop on formal aspects in security and trust
53. Gamble MT, Gamble R (2008) Isolation in design reuse. J Softw Process Improv Pract 13:145–156
54. Xie R, Gamble R (2013) An architecture for cross-cloud auditing. In: 8th Cyber security and information intelligence research workshop
55. Raj H, Nathuji R, Singh A, England P (2009) Resource management for isolation enhanced cloud services. In: Proceedings of the ACM workshop on Cloud computing security workshop
56. Bernsmed K, Jaatun M, Meland P, Undheim A (2011) Security SLAs for federated cloud services. In: Sixth international conference on availability, reliability and security
57. Handorean R, Roman G-C (2003) Secure sharing of tuple spaces in ad hoc settings. ENTCS 85(3):122–141
58. Merrick I, Wood A (2000) Coordination with scopes. In: Proceedings of the 2000 ACM symposium on applied computing
59. Sudhir A, Carriero N, Gelernter D (1986) Linda and friends. IEEE Comput 19(8):26–34
60. Mani Chandy K (1988) Parallel program design: a foundation. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA
61. Wang W, Gupta A, Wu Y (2015) Continuously delivered? Periodically updated? Never changed? Studying an open source project's releases of code, requirements, and trace matrix. In: 2015 IEEE workshop on just-in-time requirements engineering (JITRE). IEEE, pp 13–16
62. Ghezzi C, Menghi C, Sharifloo AM, Spoletini P (2014) On requirement verification for evolving Statecharts specifications. Requir Eng 19(3):231–255
63. Mahmoud A, Niu N (2014) Supporting requirements to code traceability through refactoring. Requir Eng 19(3):309–329
64. Hermoye L, Lamsweerde A, Perry D (2014) A reuse-based approach to security requirements engineering. http://users.ece.utexas.edu/~perry/work/papers/060908-LH-reuse.pdf
65. van Hermoye LA, Perry DE (2006) Attack patterns for security requirements engineering
66. Saeki M, Kaiya H (2008) Security requirements elicitation using method weaving and common criteria. In: International conference on model driven engineering languages and systems. Springer, pp 185–196
67. Vyatkin V (2013) Software engineering in industrial automation: state-of-the-art review. IEEE Trans Ind Inf 9(3):1234–1249
68. Yu Y, Franqueira VNL, Tun TT, Wieringa RJ, Nuseibeh B (2015) Automated analysis of security requirements through risk-based argumentation. J Syst Softw 106(Supplement C):102–116
69. Darimont R, Delor E, Massonet P, van Lamsweerde A (1997) GRAIL/KAOS: an environment for goal-driven requirements engineering. In: Proceedings of the 19th international conference on software engineering. ACM, pp 612–613
70. Profile EP, E-COFC public business class. ECMA Technical Report TR/781999
71. MITRE. Common attack pattern enumeration and classification (CAPEC) initiative. MITRE Corporation. http://cwe.mitre.org/. Accessed Oct 2015
72. Davies J, Woodcock J (1996) Using Z: specification, refinement and proof. Prentice Hall International Series in Computer Science. ISBN 0-13-948472-8
73. Clarkson M, Schneider F (2010) Hyperproperties. J Comput Secur 18(6):1157–1210
74. (2012) FedRamp Baseline Security Controls. www.gsa.gov/graphics/staffoffices/FedRAMP_Security_Controls_Final.zip. Accessed Oct 2015
75. (2009) Cloud security alliance, security guidance for critical areas of focus in cloud computing v3.0. https://cloudsecurityalliance.org/wp-content/uploads/2011/07/csaguide.v2.1.pdf. Accessed Oct 2015