applied sciences

MDPI

Article

State-Indexed PCI DSS Compliance: A Formalism for Path-Sensitive Valuation in Adversarial Networked Payment Systems

Fatma Yasmine Loumachi [1,*], Karim Ouazzane [1] and Anthony Phipps [1,2]

[1] Cyber Security Research Centre, London Metropolitan University, London N7 8DB, UK; tskouaz1@londonmet.ac.uk (K.O.); anthony.phipps@lloydsbanking.com (A.P.)
[2] Lloyds Banking Group, London EC2V 7HN, UK
* Correspondence: fal0167@my.londonmet.ac.uk

Abstract

Payment infrastructures are security-critical networked systems whose control status may change as attacks, remediation, and recovery unfold. In such environments, model-level PCI DSS requirement-check applicability and control satisfaction may vary across intermediate operational states. A deliberately restricted point-in-time terminal-state projection may consequently omit model-level violations represented at intermediate states during malware infection, lateral movement, segmentation failure, service disruption, or subsequent restoration. This paper introduces SCF-PCI, a formal and executable framework for path-sensitive PCI DSS valuation over adversarially evolving payment-system states. The framework defines a PCI valuation domain, separates requirement-check applicability from satisfaction, and assigns state-indexed valuations to admitted checks over reached PCI-valued states. These valuations are then lifted to execution paths, whilst endpoint loss is characterised through restoration, applicability-window closure, and domain exit. SCF-PCI is implemented in OPA Rego using OSCAL-represented PCI DSS control artefacts and evaluated across a ten-scenario analytical corpus involving malware infection, remote compromise, DDoS disruption, spyware-assisted fraud, and recovery paths. The results show that, under the instantiated predicate mappings, state-indexed valuation retains model-level violations that are omitted by the analytical terminal-state projection. The framework provides a formal proof-of-concept for state-aware PCI DSS assessment support in adversarial payment environments, without replacing conventional PCI DSS assessment or assessor judgement.

Keywords: PCI DSS; cybersecurity; payment systems; formal methods; path-sensitive analysis; adversarial networks

Check for updates

Academic Editor: Rui Araújo

Received: 13 July 2026
Revised: 8 September 2026
Accepted: 9 September 2026
Published: 3 November 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

Next-generation payment infrastructures can be characterised as distributed networked systems in which cardholder data environments (CDEs), payment terminals, authentication services, fraud-monitoring components, cloud-hosted services, APIs, and operational monitoring pipelines interact across technical and organisational boundaries. These infrastructures support availability, data exchange, and real-time payment services, but they also create security-assurance conditions in which control status may change as adversarial activity, remediation, and system transitions unfold [1]. An intrusion may alter trust relationships, weaken segmentation, disrupt monitoring, affect authentication, or expose

cardholder data (CHD), whilst later remediation may restore part of the system before an assessment point is reached. For standards-based network security, the resulting problem is not only whether a control appears satisfied at a final state but whether the standard relevant facts that made a requirement applicable, satisfied, or violated have been preserved across the operational path.

The Payment Card Industry Data Security Standard (PCI DSS) [2] provides a concrete security-standard case for this problem. PCI DSS establishes baseline requirements for protecting CHD across the payments ecosystem and operates as an industry security standard enforced through payment-brand and acquiring-bank relationships. PCI DSS v4.0.1 organises these obligations across twelve principal requirements covering network security, data protection, identity and access management, vulnerability management, monitoring, testing, and governance. Although the standard defines security objectives and validation expectations, it allows implementation flexibility, particularly where entities justify customised control approaches. The same requirement may consequently be implemented through different control designs, supporting artefacts, and assessment procedures [3]. Interpretive variation may also arise within organisations and across qualified security assessors (QSAs), including variation in how requirements are interpreted, evidenced, and remediated [4,5]. Where CHD can be linked to an identifiable individual, payment-data security may also carry data-protection significance, alongside its primary treatment under PCI DSS as a payment-security concern [5].

The technical difficulty is that PCI DSS applicability is determined by operational facts that may change during adversarial system evolution. A requirement item may depend on whether CHD or sensitive authentication data is present, whether a system stores, processes, transmits, connects to, or can affect the security of the CDE, whether segmentation supports an out-of-scope determination, or whether a particular asset or data-handling context exists. In conventional PCI DSS assessment practice, these matters are resolved through scoping analysis, evidence review, and assessor judgement. In networked payment environments, compromise may unfold across a sequence of operational states [6,7], where events expose CHD, affect trust relationships, weaken segmentation, disrupt security functions, or change the evidence available for later assessment [8]. A system initially treated as outside direct CHD handling may become relevant once it provides a route into the CDE or begins storing, processing, or transmitting payment-card account data. If the intrusion is later remediated, a point-in-time terminal-state projection may represent the system as restored whilst omitting the earlier interval in which a requirement-check was applicable and the corresponding control condition failed [9]. Applicability and violation can therefore be evaluated over the reached states of the scenario and compared with the corresponding terminal-state projection [3,10].

Network and system security research provides methods for modelling attacks, analysing adversarial progression, monitoring runtime behaviour, supporting digital forensics, and automating policy checks. These methods are valuable for representing technical events and security conditions across evolving systems, whilst compliance automation and policy-as-code provide mechanisms for executable control checking. When these mechanisms are applied to PCI DSS reasoning, however, an explicit valuation layer is needed to distinguish three different cases that are easily collapsed in a terminal-state representation, such as, for instance, a system outside the PCI valuation domain, a requirement-check that is in domain but not applicable in the current state, and a requirement-check that is applicable but not satisfied. Without this distinction, two adversarial paths may share the same final state whilst differing in whether CHD was exposed, segmentation failed, or a PCI DSS control condition was violated at an earlier state. Restoration, closure of an applicability

window, and exit from the valuation domain can then remove evidence of an intermediate model-level standards-relevant failure from the terminal-state projection.

SCF-PCI is introduced as a formal and executable standards-assurance framework for adversarially evolving payment-system states. It builds on a previously validated institutional payments ontology [8,11,12], developed through prior collaboration between London Metropolitan University and a UK Other Systemically Important Institution (O-SII), which represents the payment and banking world in which fraud scenarios unfold. SCF-PCI extends this ontology with a normative PCI DSS layer in which requirement-checks denote model-level abstractions over PCI DSS requirement items and their applicability and satisfaction conditions. The framework evaluates those checks over the states produced by the underlying situation–action model, separates valuation-domain membership, check-level applicability, and control satisfaction, and implements the resulting state-indexed valuation semantics through executable Open Policy Agent (OPA) Rego control-checking artefacts. It targets settings in which a discretised sequence of system states already exists, such as post-incident forensic reconstruction, pre-incident threat modelling, and scenario-based network security analysis. Its role is to flag potential PCI DSS-relevant control failures along an adversarial path, including intermediate failures that may be lost once remediation, scope change, or restoration has occurred, without replacing formal PCI DSS assessment, Report on Compliance (ROC) reporting, or QSA [2].

The contributions are as follows.

- It introduces SCF-PCI, a state-indexed and path-sensitive standards-assurance framework for evaluating PCI DSS requirement-check applicability and control satisfaction across adversarial payment-network states.
- It formalises the endpoint-loss problem in standards-based network security analysis and identifies restoration, closure of an applicability window, and exit from the PCI valuation domain as distinct mechanisms through which intermediate model-level PCI DSS-relevant control failures may be absent from a terminal-state projection.
- It provides an executable OPA/Rego instantiation of the valuation semantics. Selected PCI DSS provisions are represented through Open Security Controls Assessment Language (OSCAL) control artefacts and decomposed into state-evaluable applicability and satisfaction predicates, so that executable control-checking results can be tested against the formal model.
- It evaluates the approach across representative payment-network attack and cyber-fraud scenarios and demonstrates, under the instantiated predicate mappings, how state-indexed valuation retains model-level standards-relevant changes that are omitted by the analytical terminal-state projection.
- It positions SCF-PCI as an assessment-support layer for networked payment security, applicable where a discretised state sequence already exists, with an explicit boundary between SCF-PCI outputs, ROC reporting, QSA judgement, and formal PCI DSS compliance determinations.

Outline

The paper proceeds from networked payment-security problem framing to standards modelling, formal state-indexed valuation, executable control checking, and scenario-based evaluation. Section 2 reviews related work on network and system security modelling, adversarial progression, runtime monitoring, standards-based security assurance, and executable control checking. Section 3 introduces the SCF-PCI framework, including the four-level model and the normative PCI DSS layer used to represent valuation-domain membership, check-level applicability, and control satisfaction across reached states. Section 4 applies the formalism to representative payment-network attack and cyber-fraud scenarios and evaluates the exe-

cutable OPA/Rego artefacts for conformance with the defined valuation semantics. Section 5 discusses limitations, including model-local scope, dependence on discretised state sequences, and the relationship between SCF-PCI outputs, ROC reporting, and QSA judgement. Section 6 concludes the paper.

2. Literature Review

Security assurance for modern payment infrastructures sits at the intersection of standards-based control assessment and network system security analysis. CDEs are not isolated artefacts; they are composed of payment devices, applications, networks, authentication services, monitoring components, and data flows that may become relevant to PCI DSS evaluation as an adversarial scenario unfolds. PCI DSS provides the normative basis for CHD protection, whilst payment environments may pass through compromise, fraud, remediation, restoration, and forensic reconstruction before their security posture is evaluated. This review examines how prior work represents security obligations, operational states, adversarial change, and control evaluation across changing networked payment-system conditions.

The review is concept-centric in scope [13,14] and compares how prior work represents security standard obligations, requirement applicability, control satisfaction, evidence status, operational state, transition, compromise, remediation, and retention of intermediate evaluations. Work on PCI DSS assessment, machine-readable control representation, executable rules, runtime verification, process compliance, payment-fraud analysis, attack modelling, and security ontologies is considered where it contributes to standards-based security evaluation over changing payment-system states.

The terms formalisation and formalism are used in their systems-modelling sense. Formalisation refers to the process of making a domain or reasoning problem explicit through defined concepts, relations, states, transitions, and evaluation rules [15,16]. Formalism refers to the structured representation through which those definitions are expressed, analysed, and applied [17]. PCI DSS is treated as the normative source of the requirement items, whilst the analytical concern is whether prior work provides the structures needed to evaluate requirement-check applicability and control satisfaction across adversarially reached payment-system states.

The discussion proceeds from three bodies of work to the integration problem addressed by this paper. Section 2.1 reviews standards-based security assurance, with attention to PCI DSS assessment practice, machine-readable control representations, executable normative rules, and automated control checks. Section 2.2 reviews formal and execution-oriented assurance, including runtime verification, temporal compliance analysis, declarative process compliance, and conformance checking. Section 2.3 reviews adversarial models of payment-system change, including payment-fraud analysis, attack graphs, domain ontologies, situation–action models, and threat-progression models. Section 2.4 identifies the unresolved relation between a PCI DSS requirement-check, the operational state in which it is evaluated, the adversarial path through which that state is reached, and the valuation retained for that state.

2.1. Standards-Based Security Assurance and Executable Control Checks

2.1.1. PCI DSS Assessment and Attestation Practice

PCI DSS assessment remains largely documentary, evidential, and attestation-oriented. Conformance depends on scoping decisions, assessor testing procedures, control evidence, and formal reporting instruments [5]. Morse and Raval [18] place this model in its institutional context and characterise PCI DSS as a contractual mechanism among payment brands and participating entities. Practitioner and survey accounts [3,10,19,20] associate

scope definition, in-scope asset identification, and requirement applicability with organisational scoping decisions and assessor interpretation. PCI DSS v4.0.1 continues within this structure and gives more explicit attention to scope confirmation, customised approaches, and targeted risk analysis.

This literature establishes the institutional and evidential basis of PCI DSS assurance, but it gives less formal attention to the representation of requirement applicability and control satisfaction when the operational facts relevant to a requirement-check vary across compromise, remediation, restoration, and post-incident reconstruction. In adversarial payment environments, this matters since a control failure may exist at an intermediate state and then disappear from an endpoint view after the system has been restored, segmented, cleaned, or removed from scope. A standards-assurance model for such settings needs to distinguish the state in which a PCI DSS check applies, the facts that support satisfaction, and the path through which the relevant state was reached.

2.1.2. Machine-Readable and Executable Control Representations

A parallel line of work seeks to make security and compliance obligations machine-readable and operationally evaluable. NIST OSCAL provides structured representations for control catalogues, profiles, assessment plans, assessment results, and related artefacts [21,22]. Rule-based approaches encode obligations through formal logic, deontic representations, defeasible rules, and standards such as LegalRuleML, which support more precise and reproducible normative evaluation [23,24]. Automated compliance verification work also uses NLP and legal-text classification to extract, classify, or compare requirements from regulatory and contractual sources [25–27].

These approaches improve standardisation, machine-readability, and automation. Their usual unit of analysis, however, is a clause, control, document, artefact, evidence item, or predefined check. Operational facts may be supplied as inputs to the evaluation, but the mechanisms through which those facts arise, persist, or cease to hold during an adversarial payment scenario usually sit outside the control representation itself. Machine-readable standards and executable rules can support structured checking, yet PCI DSS evaluation over adversarial payment-system evolution also requires an explicit state model that connects each check to the operational facts present at the reached state.

2.2. Formal and Execution-Oriented Assurance

2.2.1. Runtime Verification and Temporal Compliance Analysis

Runtime verification treats assurance as an execution-oriented problem in which traces of states or events are checked against formal properties as observations accumulate [28,29]. Foundational applications to legal and regulatory obligations include execution-trace checks for regulatory conformance [30], logical specifications of HIPAA and GLBA privacy obligations [31], and metric first-order temporal logic for security and usage-control policies [32]. Related systems such as MonPoly demonstrate that temporal policy monitoring can be applied to real system data [33,34]. More recent work extends this line through cross-instance regulatory compliance checks over business-process event logs [35], predictive compliance analysis for process-aware information systems [36], and regulatory compliance analysis over process execution data [37].

This body of work provides formal machinery for verdicts over observed behaviour. For PCI DSS reasoning in adversarial payment infrastructures, however, the trace must be connected to the security-relevant situation represented at each state. A compromised payment terminal, infected host, exposed authentication path, altered segment, or restored asset can change the applicability and satisfaction conditions of a requirement-check. Trace-based compliance analysis supports the premise that verdicts can vary across execution,

but PCI DSS-specific valuation also requires a representation of the payment-system facts on which each requirement-check depends.

2.2.2. Declarative Process Compliance and Conformance Checking

Business-process and information-systems research gives execution-oriented assurance a process-centred form. Declarative process compliance approaches use constraint-based models and temporal logics to evaluate whether observed or specified behaviour satisfies regulatory and organisational requirements [38,39]. Conformance checking compares observed behaviour with a reference model to localise deviations and explain mismatches [40]. Compliance-by-design embeds obligations into process specifications before execution [41], while continuous-auditing research examines how control assurance can be maintained across ongoing operational activity [42].

These strands provide important concepts for executable assurance, especially the treatment of compliance as a property that can be evaluated across process execution or audit evidence. Their representational focus is usually the process model, event log, reference behaviour, or control objective. In adversarial payment settings, the relevant problem is not only whether an execution conforms to a process model but also whether the operational state reached by an attack or remediation path makes a PCI DSS requirement-check applicable and satisfied. SCF-PCI draws on the execution-oriented logic of this literature while shifting the valuation target to standards-derived checks evaluated over adversarially reached payment-system states.

2.3. Adversarial Models of Payment-System Change

2.3.1. Payment Fraud as Predictive Security Analysis

Payment-fraud research is one of the closest bodies of work on adversarial behaviour in payment environments. It usually treats fraud detection as a predictive security problem over transaction streams. Dal Pozzolo et al. [7] formalise the credit-card setting under extreme class imbalance, concept drift, and delayed supervision. Carminati et al. [9] analyse adversarial mimicry in banking fraud detection. Lunghi et al. [6] frame fraud detection through adversarial learning under real-world operational constraints. Related work addresses generative methods for class imbalance [43,44] and digital-twin approaches to credit-card fraud detection [45].

This literature is useful for establishing payment fraud as an adaptive, temporally evolving security problem. Its analytical outputs, however, are mainly predictive: labels, scores, rankings, synthetic samples, alerts, and monitoring signals. These outputs support detection and prioritisation, but they usually do not make the evolving operational context itself the object of standards-based valuation. For PCI DSS assurance, the question is not only whether a transaction, user, device, or event is suspicious, but also whether the payment-system state reached during the scenario changes the applicability or satisfaction of a security-standard check.

2.3.2. Attack Graphs and Threat-Progression Models

Attack-graph research provides a formal basis for representing attacks as structured changes in system state. Early work established that attacker capabilities, vulnerabilities, privileges, and system conditions can be represented as reachable configurations linked by transitions, which supports automatic generation and analysis of compromise paths [46]. MulVAL recasts this problem in a declarative logic-programming setting, deriving attack paths from facts about hosts, vulnerabilities, privileges, and network reachability [47]. Later work demonstrated that logical attack-graph analysis can scale to realistic enterprise

networks [48]. The cyber kill chain of Hutchins et al. [49] provides a complementary staged account of adversarial activity, from initial preparation to objective completion.

These models are central to security analysis since they represent attack progression through changing adversary capabilities and changing system conditions. They provide a strong basis for reasoning over compromise paths, but their main valuation target is usually reachability, exploitability, privilege escalation, or attack progression. PCI DSS assurance raises a related but distinct question: given a reached operational state, which standards-derived requirement-checks enter the valuation domain, which are applicable at the check level, and which are satisfied or violated in that state?

2.3.3. Domain Ontologies and Situation–Action Models

Semantic-security research extends attack and threat models through typed vocabularies for the environment under analysis. Ontology-enriched attack graphs and OWL/SWRL-based approaches [11,12,50–52] represent assets, vulnerabilities, conditions, attacker capabilities, domain relations, and inference rules within a shared semantic structure. Their contribution lies in the explicit representation of entities and relations within attack scenarios, so that operational change can be expressed through typed facts and not only through connectivity or reachability.

Within this semantic-security line, situation–action models focus on the succession of operational situations through which a security domain moves. The tradition introduced by Bataityte et al. [11] and developed in later semantic-security work [12] represents a security domain through situations, entities, conditions, events, actions, and rules. This separation keeps state descriptions, event occurrence, and state-changing actions analytically distinct. Sowinski-Mydlarz et al. [8] applies this model to banking-threat scenarios, where operational journeys are instantiated as graphs that move through normal, abnormal, corrective, and terminal situations.

This strand provides the closest representational basis for SCF-PCI. It supports explicit description of assets, data artefacts, events, conditions, and actions that may vary across a payment-security scenario. These facts do not constitute PCI DSS assessment on their own, since PCI DSS also depends on scope, evidence, testing procedures, and assessor judgement. They do, however, provide the operational state substrate on which a standards-based PCI DSS valuation layer can operate.

2.4. Gap: Path-Sensitive PCI DSS Valuation for Networked Systems

Prior work addresses PCI DSS assessment, executable control checks, temporal compliance analysis, and adversarial system modelling as largely separate problems. PCI DSS assessment defines scope, evidence, requirement applicability, findings, and remediation, while attack-modelling and situation–action approaches represent compromise paths, network attacks, and operational change. What remains under-specified is the relation between these two dimensions: how PCI DSS-relevant scope, applicability, and control-satisfaction conditions should be represented when a connected payment system changes state under attack, containment, remediation, and recovery.

In a networked payment environment, a component may become PCI-relevant not only when it directly stores, processes, or transmits cardholder data but also when it connects to, affects, exposes, disrupts, or restores parts of the cardholder-data environment. Existing approaches do not sufficiently explain how such scope-relevant relationships change across successive adversarial states, or how intermediate control failures should be retained when later remediation, re-segmentation, or restoration changes the final assessment condition.

A state-indexed and path-sensitive valuation account is therefore needed for PCI DSS reasoning in networked payment systems. The situation–action ontology provides a

suitable foundation for this account, since it represents an interconnected payment and banking environment derived from real UK banking-sector collaboration, with networked entities, services, data artefacts, events, actions, operational conditions, attack transitions, and inspectable evidence [8,11,12]. SCF-PCI addresses this integration problem by extending the situation–action ontology of Bataityte et al. [11,12] and the banking-threat model of Sowinski-Mydlarz et al. [8] with a PCI DSS valuation layer.

3. Methodology and Formal Model Construction

The methodology of the proposed formalism treats PCI DSS as the normative security standard and applies it over a semantically represented networked payment environment. SCF-PCI is constructed as a model-level valuation layer that records PCI DSS requirement-check applicability and control satisfaction across operational states reached through attack, containment, remediation, and recovery. The aim is to support standards-based network and system security analysis by preserving PCI DSS-relevant control conditions across adversarial payment-system paths.

For each instantiated PCI DSS requirement-check admitted to valuation, SCF-PCI assigns a state-indexed valuation within the typed operational state. The valuation first determines whether the check applies in the reached state, then determines whether the corresponding control predicate is satisfied within the modelled environment. This valuation is an assessment-support artefact, not a formal PCI DSS finding. It records the model-local status of a requirement-check, while formal PCI DSS assessment remains dependent on the wider assessment process, including scope confirmation, testing procedures, evidence sufficiency, source reliability, ROC reporting, and QSA professional judgement [4]. No proprietary assessment workpapers, confidential organisational data, or non-public QSA materials were used.

The inherited modelling foundation is a situation–action ontology for dynamic security-policy behaviour [11]. As illustrated in Figure 1, in its foundational form, the framework distinguishes *Situations*, *Events*, *Entities*, and *Action* relations, where events occur within situations and actions define transitions between situations.

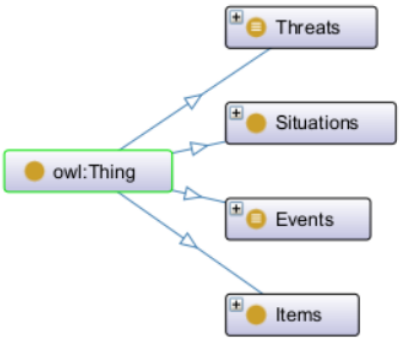

Figure 1. Ontology constructs from the original situation–action ontology developed at London Metropolitan University by the authors and collaborators in prior work [11,12].

Subsequent threat intelligence validation work operationalises this framework for banking scenarios using ontology concepts such as situations, events, threats, items, and actions [8]. The present construction adopts that banking threat intelligence vocabulary and makes *Conditions* explicit as state predicates qualifying items within situations, since PCI DSS requirement-check valuation depends on whether particular security-relevant conditions hold or fail in a reached operational state.

A *Situation* (Figure 2) denotes a partial, evidence-grounded characterisation of the operational environment at a point of assessment.

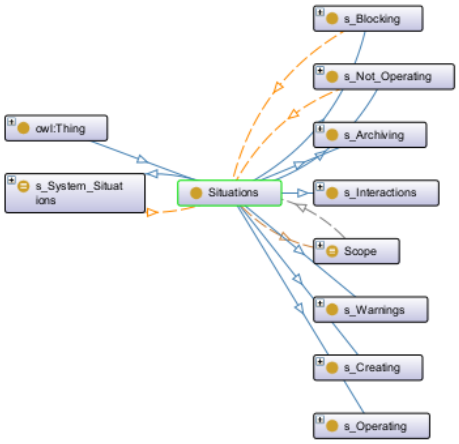

Figure 2. Situation subclasses.

An *Event* (Figure 3) denotes an asynchronous occurrence observable within a situation. An *Action* denotes a controlled transition between situations and constitutes the sole mechanism through which state change is expressed.

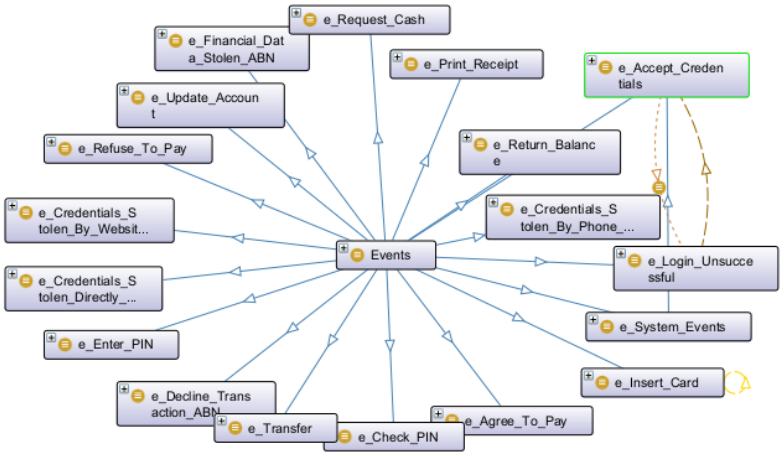

Figure 3. Event subclasses.

Items (Figure 4) denote persistent operational objects within the environment, such as ATMs, cards, authentication services, and data artefacts.

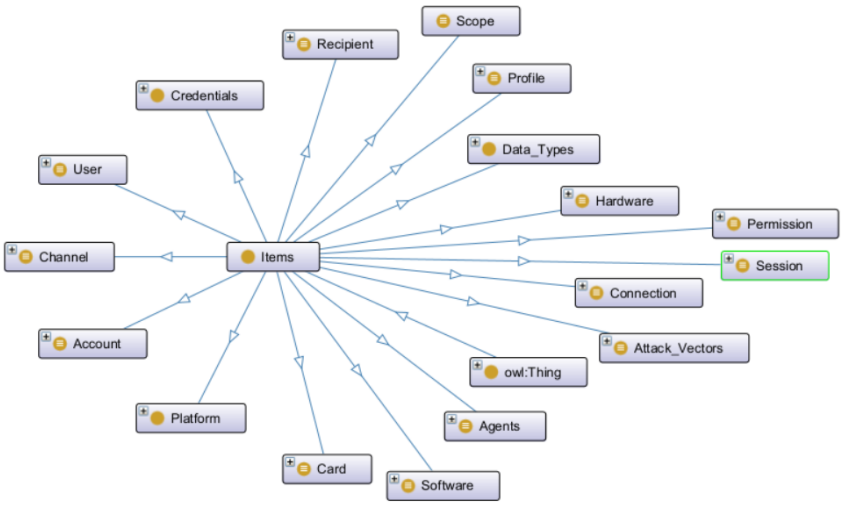

Figure 4. Item subclasses.

Conditions denote qualitative, security-relevant properties that hold for items within a situation, including patch state, encryption state, credential exposure, logging coverage, and control activation. *Threats* (Figure 5) denote adversarial capabilities or modes of compromise that may give rise to events and influence state transitions.

The original framework is organised into three distinct levels as shown in Figure 6. The *Ontological Level* defines the domain vocabulary and admissible relations. The *Logical Level* specifies behavioural and threat rules governing transitions between situations. The *Analytical Level* derives executable scenarios and transition systems induced by those rules.

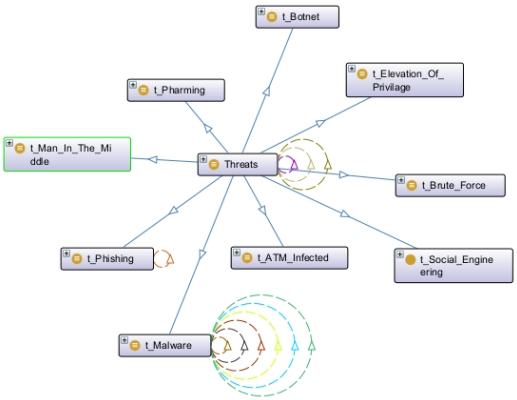

Figure 5. Threat subclasses.

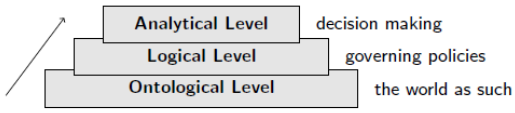

Figure 6. A three-level semantic framework [11].

We introduce an additional Normative Compliance Level, illustrated in Figure 7, with its construction informed by the representational requirements identified in Section 2.

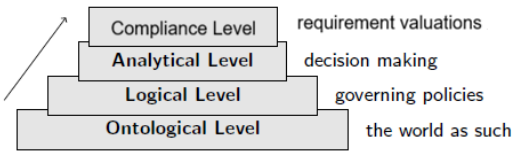

Figure 7. A four-level semantic framework.

This level leaves the ontology and behavioural dynamics unchanged and assigns valuation judgements by evaluating PCI DSS requirement-check predicates over the situations produced by the lower levels, without altering operational semantics.

3.1. Ontological Level and World Model

The world model Σ is a description logic knowledge base that represents the operational environment as a state-centric semantic structure:

$$\Sigma = (\mathcal{T}Box, RBox, \mathcal{A}Box)$$

where the $\mathcal{T}Box$ defines the domain vocabulary and its semantics through concept hierarchies, disjointness constraints, and typing rules; the *RBox* defines admissible structural relationships between concepts through role definitions and domain/range constraints; and the $\mathcal{A}Box$ contains assertions over individuals representing instantiated situations, events, items, conditions, threats, and actions.

The domain is organised around five pairwise disjoint top-level concepts:

$$\text{DisjointClasses}(\mathit{Situation}, \mathit{Event}, \mathit{Item}, \mathit{Condition}, \mathit{Threat})$$

This separation keeps distinct the state being described, the events observed within that state, the operational objects present in the environment, the security-relevant properties attached to those objects, and the adversarial conditions represented in the scenario.

Structural relationships are captured through the following roles:

- `hasItem(Situation, Item);`
- `hasCondition(Situation, Condition);`
- `about(Condition, Item);`
- `occur_in(Event, Situation);`
- `Action(Situation, Situation).`

Conditions are interpreted relationally so that a condition instance holds in a situation and qualifies a specific item in that situation. This allows security-relevant properties to be represented as state-local facts and to be carried across successor situations unless altered by an action. State evolution is treated as *epistemic*, in that a situation represents the best supported characterisation of the world given the admissible evidence available at a point of assessment. In operational banking environments, system state is known through logs, alerts, configuration artefacts, and recorded events. The catalogue of situations and transitions is treated as stable to provide a controlled semantic frame within which PCI DSS valuation semantics can be formalised and evaluated.

3.2. Logical Level and Behavioural Rules

Over the world model Σ, a set of behavioural and threat rules R_{dyn} governs admissible world dynamics. These rules specify how operational situations evolve in response to system behaviour and adversarial activity. They operate strictly over the concepts and roles defined in the *TBox* and *RBox*. Their role is behavioural, not normative. They define admissible state change, not PCI DSS requirement meaning or compliance valuation.

The dynamic knowledge base is defined as

$$K_{\text{dyn}} = (\Sigma, R_{\text{dyn}})$$

where R_{dyn} is a set of Horn-style rules of the form $\mathit{body}(x) \rightarrow \mathit{head}(x)$, with atoms drawn exclusively from the ontological vocabulary. In line with prior work, rules in R_{dyn} are expressed in SWRL and constitute the logical policy level of the model.

Each rule captures a local causal relation within Σ that specifies how the occurrence of an event or the presence of a condition in a situation gives rise to a successor situation in which the distribution of conditions across items may be altered. The rule set is deliberately fixed to secure a controlled semantic regime, and any future introduction of adaptive rule induction must preserve the causal structure defined by R_{dyn}.

3.3. Analytical Level and Scenario Transition System

The analytical level [8] makes the behavioural semantics of the logical level explicit as a labelled transition system $\mathcal{M}$ suitable for scenario reasoning:

$$\mathcal{M} = (S, A, \rightarrow)$$

where S is the set of situation individuals, A is the set of action labels, and $\rightarrow \subseteq S \times A \times S$ is the transition relation. A transition $(s, a, s') \in \rightarrow$ holds if the action assertion $\texttt{Action}_a(s, s')$ is entailed by K_{dyn}.

An adversarial scenario path ρ is any finite run:

$$\rho = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{n-1}} s_n$$

At this level, ρ is purely behavioural, recording how the world may evolve according to R_{dyn}. The transition system $\mathcal{M}$ is characterised by four structural properties. *Reachability* confirms that all situations arise from the initial state s_0. *Dead-End* identification establishes terminal situations with no outgoing transitions, representing final outcomes such as recovery or loss. *Cycle* analysis confirms that the scenario reflects directed progression from vulnerability through compromise to resolution, with local cycles marking repeated admissible behaviour within a single situation. *Rule-Coverage* confirms that each rule in R_{dyn} fires in at least one admissible context, establishing that the behavioural specification is neither partial nor redundant.

In the intelligence graph in Figure 8, terminal situations are S_5 and S_{32}, each characterised by the absence of outgoing action transitions. A local cycle is present at S_2 through the self-transition a_{22}. Apart from this local cycle, the remaining situations form a directed acyclic progression from S_0 towards the terminal outcomes.

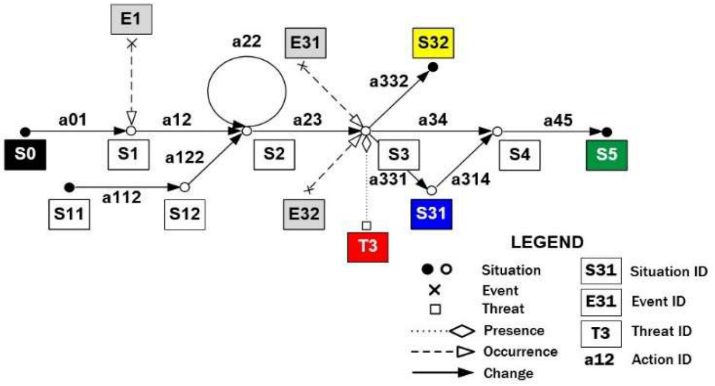

Figure 8. Intelligence graph for modelling transactions under threat [8].

Figure 9 illustrates one instantiated scenario produced by the inherited situation–action model.

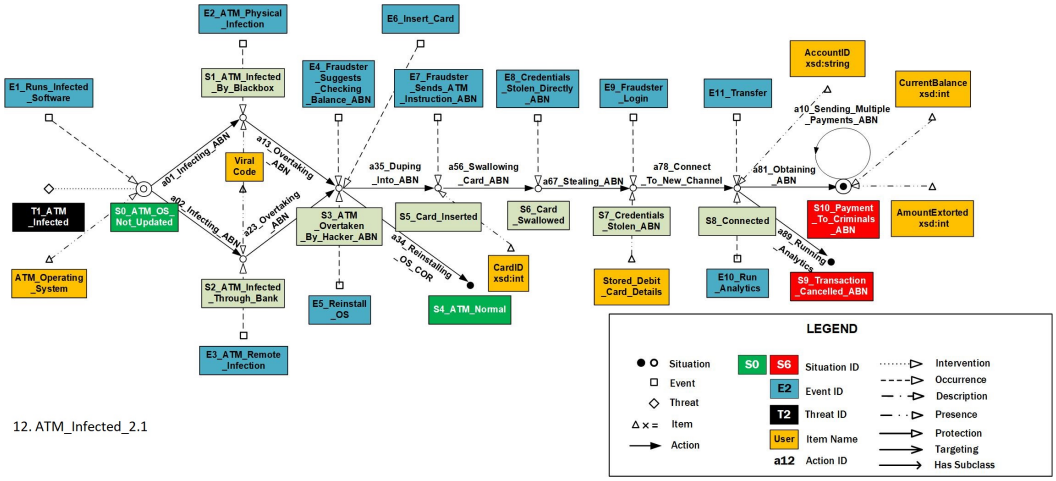

Figure 9. ATM infection scenario [8].

The ATM infection in Figure 9 contains alternative physical and remote compromise branches, corrective recovery, intervention, and criminal continuation. It is used later as one verification case, since its paths expose how requirement-check valuations may persist,

be repaired, or remain visible across adversarial progression. At the methodological level, the important point is the general construction. Scenario states are represented as typed situations, transitions are induced by behavioural rules, and the resulting paths provide the operational states over which PCI DSS requirement-check predicates are evaluated.

3.4. Normative Compliance Level and State-Indexed Valuation

The normative level adds an interpretive layer over the situations reached in $\mathcal{M}$. It does not introduce new operational transitions and does not modify the behavioural rules in R_{dyn}. Instead, it binds PCI DSS requirement-checks to reached situations and resolves applicability separately from satisfaction. This distinguishes the state in which a check first applies, the state in which its associated control condition fails, and the state in which that condition is restored. Requirement-check predicates are expressed over ontologically typed concepts defined in the *TBox* of Σ, which provides both the world model on which the evaluated states are defined and the typed vocabulary in which control conditions are expressed.

3.4.1. State Snapshot

For each reached situation $s \in S$, let

$$\pi(s) := \langle O_s, \Gamma_s \rangle$$

denote the state snapshot of s, where O_s is the set of ontological items present in s and Γ_s is the set of state conditions licensed for s. A condition is licensed only where it is supported by the semantic content of the reached situation, explicitly represented items, threats, or events, transition semantics governing persistence or alteration, or an explicit model-local evaluation input with recorded provenance. Predicates without such support are excluded from the reconstructed snapshot. Events and actions in $\mathcal{M}$ record how the system moves from one situation to another, while requirement-check interpretation is assigned to the reached situation itself.

Condition Licensing

Condition licensing specifies the admissibility rule for membership in Γ_s. A predicate is included only where it has a typed scenario or transition basis, or is supplied as an explicit model-local evaluation input with recorded provenance. Predicates without such a basis are excluded from the reconstructed snapshot and play no role in valuation.

PCI Evaluation Context

Before requirement-check applicability is evaluated, SCF-PCI requires a licensed PCI evaluation context for the reached situation. This context records only whether the ontology-derived state contains a represented basis for PCI DSS valuation, such as CHD handling, account, session, or transaction context, a CDE system component, a system that can affect the security of the CDE, or payment journey infrastructure represented as security relevant to the CDE or transaction context. Where this basis is absent or undetermined, the reached situation is not valued by SCF-PCI unless additional licensed inputs are introduced.

We therefore define the PCI valuation domain:

$$D_{\text{PCI}} = \{s \in S \mid \text{PCIContext}(\pi(s)) = \texttt{true}\}.$$

Requirement-check valuation is defined only for states in D_{PCI}; states outside this domain remain unvalued unless the missing PCI evaluation context is supplied as an additional licensed input.

3.4.2. Requirement-Check Applicability and Satisfaction

For each reached situation $s \in D_{\mathrm{PCI}}$ and each principal requirement $r \in \mathrm{Req}$, let $\mathrm{Checks}(r)$ denote the set of instantiated requirement-checks associated with r in this work. A requirement-check is a state-evaluable formal unit derived from PCI DSS requirement text. It is not an ontological item in Σ, and it is not itself the mapping step by which requirement text is translated into formal predicates. That mapping produces the predicates by which each check is evaluated.

For each check $c \in \mathrm{Checks}(r)$, two predicates are defined over reached situations in D_{PCI}:

$$\mathrm{App}_{r,c}(\pi(s))$$

states that requirement-check c associated with principal-requirement r is applicable in situation s, and

$$\mathrm{Sat}_{r,c}(\pi(s))$$

states that, where applicable, the satisfaction condition associated with check c holds in situation s.

The predicates $\mathrm{App}_{r,c}(\pi(s))$ and $\mathrm{Sat}_{r,c}(\pi(s))$ are defined only for states in D_{PCI}, while states outside D_{PCI} lie beyond the valuation domain and receive no requirement-check valuation unless the missing PCI evaluation context is supplied as an additional licensed input.

Applicability concerns the operational conditions under which a check attaches, for which evaluation is defined only where the licensed state conditions corresponding to its applicability criteria are present in $\pi(s)$. Satisfaction concerns whether the associated control condition holds in that reached state. For an applicable check, satisfaction is evaluated only where the required licensed evidence is resolved; otherwise, the state–check pair is retained as unresolved by Algorithm A2 and does not enter check-level valuation. This prevents unresolved evidence from being interpreted as control failure.

Example

Consider the corrective branch of the ATM infection scenario, $S0 \rightarrow S2 \rightarrow S3 \rightarrow S4$, introduced in Figure 9 and evaluated in detail in Section 4. At $S0$, the state snapshot contains `ATM_OS`, the state lies within D_{PCI}, and *patch-status* = false. For the instantiated PCI DSS 6.3.3 patching check, the presence of `ATM_OS` establishes applicability, while the represented patch status does not satisfy the associated control condition, yielding ViolatedInState. At the corrective state $S4$, `ATM_OS` remains represented and the state remains within D_{PCI}, but *patch-status* = true. The check remains applicable, while its satisfaction condition now holds, producing SatisfiedInState. Taken together, these states illustrate the evaluation sequence developed below: construction of the state snapshot, admission to the PCI valuation domain, resolution of applicability, evaluation of satisfaction, and assignment of the corresponding state-indexed valuation. Section 3.4.5 subsequently extends these state-level valuations to execution paths. A detailed protocol documenting the valuation procedure and the instantiation of the corresponding requirement-checks is provided in the accompanying dataset [53] in Section 4.3, along with the state catalogues, applicability matrices, valuation matrices, path metrics, reference checks, and sensitivity checks for all ten evaluated scenarios.

3.4.3. Check-Level Valuation

The three-valued semantics below apply after trace preparation has separated applicable state–check pairs with unresolved satisfaction evidence. Such pairs remain in the `UnresolvedEvidenceMap` and do not enter $V_{r,c}$; no additional valuation outcome is introduced.

For each reached situation $s \in D_{\mathrm{PCI}}$, each principal-requirement $r \in \mathrm{Req}$, and each requirement-check $c \in \mathrm{Checks}(r)$ admitted to check-level valuation, define the state-indexed valuation

$$V_{r,c}(s) \in \{\mathsf{NotApplicable}, \mathsf{SatisfiedInState}, \mathsf{ViolatedInState}\}.$$

The valuation function is defined by

$$V_{r,c}(s) := \begin{cases} \mathsf{NotApplicable}, & \text{if } \neg \mathrm{App}_{r,c}(\pi(s)), \\ \mathsf{SatisfiedInState}, & \text{if } \mathrm{App}_{r,c}(\pi(s)) \wedge \mathrm{Sat}_{r,c}(\pi(s)), \\ \mathsf{ViolatedInState}, & \text{if } \mathrm{App}_{r,c}(\pi(s)) \wedge \neg \mathrm{Sat}_{r,c}(\pi(s)). \end{cases}$$

Each requirement-check admitted to check-level valuation receives exactly one valuation: NotApplicable, SatisfiedInState, or ViolatedInState. The value NotApplicable denotes that the corresponding requirement-check does not apply within a PCI-relevant state; states outside D_{PCI} receive no valuation.

Principal-Requirement Valuation Profile

The valuation associated with principal-requirement r at situation $s \in D_{\mathrm{PCI}}$ is the family of assigned check-level valuations under that requirement:

$$\mathrm{Prof}_r(s) := \left\langle V_{r,c}(s) \right\rangle_{\substack{c \in \mathrm{Checks}(r) \\ V_{r,c}(s) \text{ defined}}}.$$

This profile is the native requirement-level object in SCF-PCI. Where a coarser summary is sufficient for downstream analysis, a derived set-valued summary may be defined:

$$\mathrm{Sum}_r(s) := \{V_{r,c}(s) \mid c \in \mathrm{Checks}(r),\ V_{r,c}(s) \text{ defined}\}.$$

Different checks under the same principal-requirement may evaluate differently at the same reached situation, so $\mathrm{Sum}_r(s)$ need not be singleton. Where downstream path analysis requires a single requirement-level status, an internal collapse function may be introduced:

$$C_r(s) := \mathrm{Collapse}(\mathrm{Prof}_r(s)).$$

$C_r(s)$ is not a native PCI DSS semantic object. It is an internal analytical convenience, applied only where a single derived status is needed.

3.4.4. Formal Properties of the Check-Level Semantics

Lemma 1 (Check-level totality)**.** *For every $r \in \mathrm{Req}$, every $c \in \mathrm{Checks}(r)$, and every reached situation $s \in D_{\mathrm{PCI}}$ admitted to check-level valuation, the valuation $V_{r,c}(s)$ is defined and returns exactly one element of*

$$\{\mathsf{NotApplicable}, \mathsf{SatisfiedInState}, \mathsf{ViolatedInState}\}.$$

Consequently, the profile $\mathrm{Prof}_r(s)$ is well-defined, and the derived summary $\mathrm{Sum}_r(s)$ may contain more than one valuation.

Proof. For a state–check pair admitted to valuation in D_{PCI}, the predicates $\mathrm{App}_{r,c}(\pi(s))$ and $\mathrm{Sat}_{r,c}(\pi(s))$ are Boolean. Exactly one of three cases holds: the check is not applicable in s; it is applicable and satisfied in s; or it is applicable and not satisfied in s. These cases are mutually exclusive and jointly exhaustive, so $V_{r,c}(s)$ is total and single-valued. The profile $\mathrm{Prof}_r(s)$ is the family of defined valuations under $\mathrm{Checks}(r)$. Since distinct checks may take different values, $\mathrm{Sum}_r(s)$ need not be singleton. □

Lemma 2 (Evaluation order). *For every reached situation s and every check (r, c):* (i) $V_{r,c}(s)$ *is assigned only if* $\mathrm{PCIContext}(\pi(s)) = \mathsf{true}$, *that is,* $s \in D_{\mathrm{PCI}}$; (ii) *the satisfaction predicate* $\mathrm{Sat}_{r,c}$ *is consulted only where* $\mathrm{App}_{r,c}(\pi(s))$ *holds; and* (iii) $V_{r,c}(s) = \mathsf{ViolatedInState}$ *implies* $\mathrm{App}_{r,c}(\pi(s))$.

Proof. Claim (i) follows from the definition of the valuation domain

$$D_{\mathrm{PCI}} = \{s \in S \mid \mathrm{PCIContext}(\pi(s)) = \mathsf{true}\},$$

since states outside D_{PCI} receive no check-level valuation. Claims (ii) and (iii) follow from the case definition of $V_{r,c}$: the predicate $\mathrm{Sat}_{r,c}$ appears only in the two branches guarded by $\mathrm{App}_{r,c}(\pi(s))$, and the $\mathsf{ViolatedInState}$ branch is guarded by $\mathrm{App}_{r,c}(\pi(s)) \wedge \neg\mathrm{Sat}_{r,c}(\pi(s))$. Since unresolved applicable pairs are screened before this valuation step, a failed satisfaction condition is recorded only where the licensed evidence resolves it as false. □

The check-level semantics above determine the valuation assigned at a single reached situation. The following definitions lift this valuation to execution paths, so that violations can be compared at the endpoint and across the full reached state sequence.

3.4.5. Path-Sensitive Retention and Endpoint Loss

For path-level comparison, let σ denote a path-selection function over the transition system $\mathcal{M}$, and let

$$\rho^{\sigma} = \langle s_0, \ldots, s_T \rangle$$

denote the finite executed path selected by σ. SCF-PCI distinguishes violations observed at the endpoint from violations observed across the path. Let

$$V_{\mathrm{path}}(\rho^{\sigma}) = \left\{ (r, c) \;\middle|\; \begin{array}{l} \exists s_i \in \rho^{\sigma} \cap D_{\mathrm{PCI}} \\ \text{such that } V_{r,c}(s_i) \text{ is defined and} \\ V_{r,c}(s_i) = \mathsf{ViolatedInState} \end{array} \right\}.$$

denote the set of requirement-checks violated at any PCI-valued state along the path, and let

$$V_{\mathrm{end}}(\rho^{\sigma}) = \begin{cases} \{(r, c) \mid V_{r,c}(s_T) \text{ is defined and } V_{r,c}(s_T) = \mathsf{ViolatedInState}\} \\ \qquad \text{if } s_T \in D_{\mathrm{PCI}}, \\ \emptyset \\ \qquad \text{if } s_T \notin D_{\mathrm{PCI}}. \end{cases}$$

denote the set of requirement-checks violated at the endpoint. Endpoint loss is then defined as

$$V_{\mathrm{miss}}(\rho^{\sigma}) = \Big\{ (r, c) \in V_{\mathrm{path}}(\rho^{\sigma}) \setminus V_{\mathrm{end}}(\rho^{\sigma}) \;\Big|\; s_T \notin D_{\mathrm{PCI}} \;\vee\; V_{r,c}(s_T) \text{ is defined} \Big\}.$$

Thus, V_{miss} is a derived object of the formal construction over evidence-resolved valuations: once a path contains a requirement-check violation that is absent from its resolved terminal-state valuation, or the endpoint leaves D_{PCI}, membership follows from the definition. The scenario evaluation subsequently instantiates this relation across different transition structures and endpoint conditions. In this comparison, V_{end} serves as an analytical terminal-state projection; conventional PCI DSS assessment remains the broader evidential and judgement-based process described elsewhere in the paper.

Proposition 1 (Endpoint soundness). *For every executed path* $\rho^{\sigma} = \langle s_0, \ldots, s_T \rangle$, $V_{\mathrm{end}}(\rho^{\sigma}) \subseteq V_{\mathrm{path}}(\rho^{\sigma})$.

Proof. If $(r,c) \in V_{\text{end}}(\rho^\sigma)$, then $s_T \in D_{\text{PCI}}$ and $V_{r,c}(s_T) = \mathsf{ViolatedInState}$. Since s_T is a reached state of ρ^σ lying in D_{PCI}, it is one of the index states of the union defining $V_{\text{path}}(\rho^\sigma)$. Hence, $(r,c) \in V_{\text{path}}(\rho^\sigma)$. If $s_T \notin D_{\text{PCI}}$, then $V_{\text{end}}(\rho^\sigma) = \emptyset$, and the inclusion holds trivially. □

Theorem 1 (Characterisation of endpoint loss)**.** *Let $\rho^\sigma = \langle s_0, \ldots, s_T \rangle$ be an executed path with endpoint s_T, and let (r,c) be such that $V_{r,c}(s_T)$ is defined whenever $s_T \in D_{\text{PCI}}$. Then, $(r,c) \in V_{\text{miss}}(\rho^\sigma)$ if and only if there exists a reached state $s_i \in \rho^\sigma \cap D_{\text{PCI}}$ with $V_{r,c}(s_i) = \mathsf{ViolatedInState}$ and exactly one of the following holds at the endpoint:*

(a) restoration: $s_T \in D_{\text{PCI}}$ and $V_{r,c}(s_T) = \mathsf{SatisfiedInState}$;
(b) applicability-window closure: $s_T \in D_{\text{PCI}}$ and $V_{r,c}(s_T) = \mathsf{NotApplicable}$;
(c) domain exit: $s_T \notin D_{\text{PCI}}$.

Cases (a)–(c) are mutually exclusive and exhaustive.

Proof. By definition,

$$V_{\text{miss}}(\rho^\sigma) \subseteq V_{\text{path}}(\rho^\sigma) \setminus V_{\text{end}}(\rho^\sigma).$$

Membership in $V_{\text{path}}(\rho^\sigma)$ states exactly that there is a reached state $s_i \in \rho^\sigma \cap D_{\text{PCI}}$ with $V_{r,c}(s_i) = \mathsf{ViolatedInState}$. By the definition of V_{miss}, either $s_T \notin D_{\text{PCI}}$, which gives case (c), or $s_T \in D_{\text{PCI}}$ and $V_{r,c}(s_T)$ is defined but is not $\mathsf{ViolatedInState}$. In the latter case, Lemma 1 gives

$$V_{r,c}(s_T) \in \{\mathsf{NotApplicable}, \mathsf{SatisfiedInState}\},$$

which yields case (b) or case (a), respectively. The three endpoint conditions on s_T are pairwise incompatible and cover all possibilities, so the cases are mutually exclusive and exhaustive. □

Theorem 1 partitions the derived endpoint-loss relation into three mutually exclusive endpoint mechanisms: restoration, applicability-window closure, and domain exit. The controlled reference cases include the domain-exit case by withdrawing the PCI evaluation context from the endpoint state, thereby placing that state outside D_{PCI} and preventing check-level valuation at the endpoint.

Implementation Note

When $C_r(s)$ is used, its ordering and resolution rules are properties of SCF-PCI, not PCI DSS reporting semantics. Given explicit licensed state conditions in $\pi(s)$ and a licensed PCI evaluation context, the valuation functions $V_{r,c}(s)$ are computable over the reached situations for the instantiated requirement-checks. Section 4.7 checks that the executable instantiation conforms to the valuation semantics by producing well-defined principal-requirement profiles over execution paths inside the PCI valuation domain.

SCF-PCI does not model PCI DSS assessment routes, including the defined approach, customised approach, targeted risk analysis, or compensating control validation, as separate valuation outcomes. These mechanisms concern how a satisfaction predicate may be justified within an assessment process. Where a customised or compensating-control basis is represented as a licensed input to the model, it may support $\text{Sat}_{r,c}(\pi(s))$; where such a basis is not represented, SCF-PCI does not infer it. The valuation function therefore remains assessment-supportive and does not substitute for PCI DSS assessment.

4. Model-Local Evaluation and Executable Conformance Testing

SCF-PCI defines a state-indexed valuation layer for standards-based networked payment-system security. Its core function is to assign PCI DSS requirement-check valua-

tions to operational states reached through adversarial payment-system scenarios and to preserve those valuations across the scenario path. The evaluation in this section checks whether the executable instantiation conforms to the valuation semantics defined in Section 3.4.3 and to the path-level retention definitions introduced in Section 3.4.5. The purpose is to show that intermediate requirement-check violations remain visible in the state-indexed trace even where they are absent from a point-in-time endpoint view of the same scenario.

SCF-PCI evaluates PCI DSS requirement-checks over the typed items, licensed conditions, and PCI evaluation context represented in the state snapshot $\pi(s) = \langle O_s, \Gamma_s \rangle$, so the evaluation remains model-local in scope. Unrepresented operational facts, live telemetry, organisational evidence, assessor judgement, and formal PCI DSS scope determinations are not inferred by the framework. They may affect valuation only where introduced as explicit licensed inputs. Scope follows from the ontology-based construction of SCF-PCI, where payment-system behaviour is represented through situations, items, conditions, events, actions, and threats before PCI DSS predicates are applied.

This section describes the executable substrate and scenario basis used for the evaluation, reports state-indexed valuation results, compares path-level valuation with a point-in-time endpoint baseline, and checks the OPA Rego implementation against selected reference cases.

4.1. Conformance Testing Procedure

Conformance testing determines whether the executable instantiation conforms to the semantic properties defined by the formal model across ordered valuation, state-indexed valuation, and path-sensitive retention. Preservation of these behaviours demonstrates that PCI DSS requirement-check outcomes are assigned to reached operational situations while intermediate violations are retained along adversarial execution paths.

The first condition, ordered valuation, requires that a situation is admitted to the valuation domain before any requirement-check outcome is determined, that applicability is resolved before satisfaction, and that control failures are recorded only when the corresponding check is applicable:

$$s \in D_{\mathrm{PCI}} \quad \Longrightarrow \quad V_{r,c}(s) = f\Big(\mathrm{App}_{r,c}(\pi(s)), \mathrm{Sat}_{r,c}(\pi(s))\Big).$$

Second, state-indexed valuation is checked by inspecting each reached situation inside D_{PCI} to confirm that applicability and satisfaction are resolved within the same state snapshot, that principal-requirement profiles are assembled from check-level valuations, and that valuation changes occur only where the represented state changes. Third, path-sensitive retention is checked by applying the path-level definitions of V_{path}, V_{end}, and V_{miss} defined in Section 3.4.5 to compare the state-indexed trace with a point-in-time endpoint baseline.

These checks are reference-case conformance tests. They assess whether the executable output is consistent with the formal valuation rules defined in this paper, but they do not amount to a machine-checked proof that the implementation satisfies the full semantics.

4.2. Executable Substrate and PCI DSS Representation

The valuation procedure defined in Section 3.4.3 is implemented using Open Policy Agent (OPA) Rego to support executable control checking over structured inputs. OPA Rego is used as an execution substrate as it separates declarative rules from application logic and provides a rule language for expressing PCI-context, applicability, and satisfaction predicates. The engineering realisation of SCF-PCI is algorithmic, so the same formal valuation semantics could, in principle, be realised in other rule-based or programmatic environments.

4.2.1. Machine-Readable Representation of PCI DSS Requirements

The executable valuation procedure requires selected PCI DSS provisions to be available in a structured form before they can be decomposed into state-evaluable predicates. PCI DSS v4.0.1 expresses its requirements as normative prose, with applicability conditions, control statements, testing expectations, and assessment-relevant components distributed across the standard and not exposed as executable objects. A structured intermediary is therefore needed to preserve the link between the authoritative requirement text and the predicates evaluated by SCF-PCI. OSCAL serves this role as a standardised, machine-readable schema for representing control catalogues and assessment-related artefacts [54].

PCI DSS is not natively issued in OSCAL form. For this study, a publicly available GitHub. repository [55] containing a PCI DSS v4.0.1 representation in OSCAL was adopted as the normative substrate. The dataset was checked against the public PCI DSS requirement text to confirm that its identifiers and statements corresponded to the provisions used in the instantiated checks.

The authoritative source of PCI DSS obligations remains the standard itself. The OSCAL representation is used only as an intermediary through which selected provisions are organised before being decomposed into requirement-checks, expressed as applicability and satisfaction predicates over $\pi(s)$, and supplied to the OPA Rego valuation procedure.

4.2.2. Execution Workflow and Computational Requirements

The executable realisation processes each scenario through a finite rule-based evaluation sequence. Reached situations are materialised as typed state snapshots, passed through the PCI evaluation-context gate, and evaluated against the instantiated applicability and satisfaction predicates using OPA/Rego. The resulting check-level valuations are retained across the selected execution paths to derive V_{path}, V_{end}, and V_{miss}. Execution requires the OPA runtime over the structured scenario and control inputs and uses standard CPU and memory resources. The executable instantiation is used to expose and test the realisation of the defined valuation semantics, and execution time was therefore not treated as a performance measure in this study. Appendix A provides the detailed correspondence between the four formal levels and their typed executable artefacts.

4.3. Evaluation Scope and Scenario Basis

To proceed, we apply the conformance testing procedure and protocol defined above to scenario structures induced from the previously validated banking-threat ontology and its associated modelling methodology [8,11,12]. This provides the represented operational world over which SCF-PCI is evaluated as a state-indexed PCI DSS valuation layer.

4.3.1. Scenario Selection and Representativeness

The scenarios were selected to exercise the valuation semantics across distinct forms of payment-system evolution, including malware infection, remote compromise, service disruption, spyware-assisted fraud, criminal continuation, and corrective recovery. Their representativeness is analytical, reflecting different combinations of operational state change, PCI evaluation-context continuity, requirement-check applicability, control failure, restoration, applicability-window closure, and domain exit. The selection is grounded in the representational basis established in Section 2.3, where the situation–action model provides a typed payment and banking world in which assets, conditions, events, actions, and adversarial transitions can be represented across successive operational states. The resulting corpus supports evaluation across distinct state and path structures and is not intended as an exhaustive taxonomy of payment-security threats.

The main text reports one scenario (ATM infection) in full state by state detail and a contrasting scenario (login-with-spyware Figure 10) in summary form through the cross-scenario metrics (Section 4.6) and the reference cases (Section 4.7). The remaining eight scenarios are processed through the same executable pipeline, and the full per-state valuation traces and path-level metrics for all ten scenarios are provided in the accompanying dataset [53].

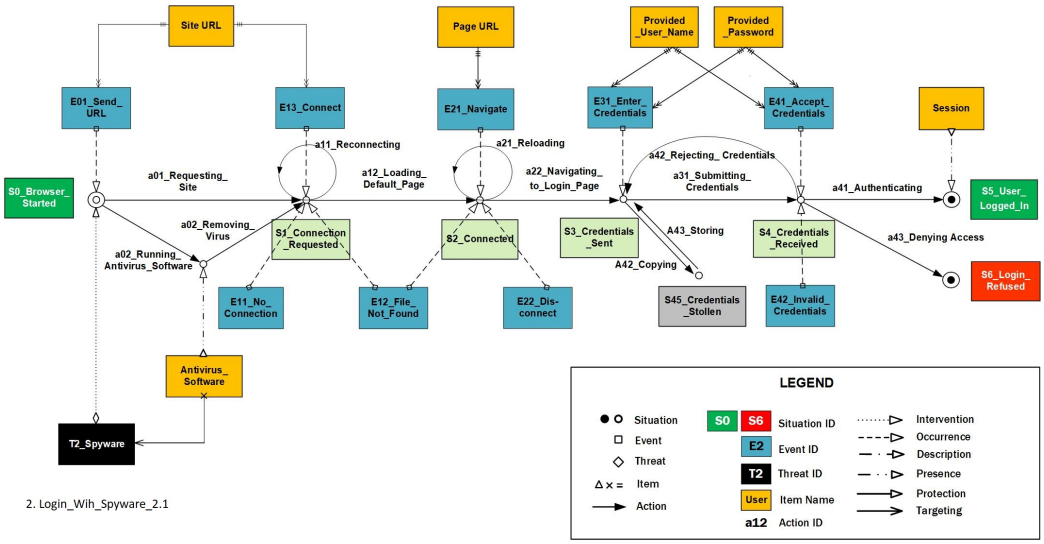

Figure 10. Login-with-spyware scenario [8].

4.3.2. Scenario 1: ATM Infection

The ATM infection scenario is used as the first detailed evaluation case. Its behavioural construction was introduced in Section 3.3. For evaluation, we consider the finite paths induced by the common transition system M and treat the repeated payment transition at S_{10} as a cyclic continuation. The graph contains two terminal finite paths, one criminal path reaching a non-terminal cyclic state, and an infinite run obtained by repeatedly applying the self-loop at S_{10}.

For consistency across path comparison, the finite paths reported in the main text use the remote infection branch through S_2:

$$S_0 \xrightarrow{a_{02}} S_2 \xrightarrow{a_{23}} S_3 \xrightarrow{a_{34}} S_4(\text{corrective}),$$

$$S_0 \xrightarrow{a_{02}} S_2 \xrightarrow{a_{23}} S_3 \xrightarrow{a_{35}} S_5 \xrightarrow{a_{56}} S_6 \xrightarrow{a_{67}} S_7 \xrightarrow{a_{78}} S_8(\text{cancelled}),$$

$$S_0 \xrightarrow{a_{02}} S_2 \xrightarrow{a_{23}} S_3 \xrightarrow{a_{35}} S_5 \xrightarrow{a_{56}} S_6 \xrightarrow{a_{67}} S_7 \xrightarrow{a_{79}} S_9 \xrightarrow{a_{91}} S_{10}(\text{criminal}).$$

The cyclic continuation is represented by

$$\rho_{\text{loop}}^{(k)} = \rho_{\text{criminal}} \cdot \left(S_{10} \xrightarrow{a_{10}} S_{10}\right)^k, \quad k \geq 1,$$

with the corresponding infinite run obtained as $k \to \infty$. In the executable evaluation, cyclic behaviour is evaluated through bounded prefixes determined by $N_{\max}$. The repeated-payment self-loop is retained as part of the behavioural semantics of the source graph. In the fixed evaluation subset, repeated execution of a_{10} does not introduce a new typed item or licensed condition. It may increase the amount or count of extortion events, but it does not change the applicability or satisfaction conditions of the instantiated checks. The loop is therefore valuation-invariant for this experiment.

The physical infection branch through S_1 is structurally symmetric to the remote infection branch up to S_3 and produces the same downstream valuation behaviour for the fixed evaluation subset used in this scenario. Although S_1 is shown in the state and

valuation tables, it is not part of the three reported path prefixes, which use the remote infection branch through S_2 for comparability. S_1 is retained to preserve the reachable-state structure of $\mathcal{M}$ and to confirm that the physical infection branch induces the same fixed-subset valuation behaviour as the remote infection branch up to S_3.

Requirement-check application protocol. The ATM scenario evaluates a fixed subset of PCI DSS checks selected from the candidate catalogue for this graph. The subset is limited to checks whose applicability and satisfaction can be resolved from the model-local state snapshots and licensed control-evidence conditions used in the scenario: (i) PCI 6.3.3 patching of the ATM operating system; (ii) PCI 5.2.1 anti-malware deployment for the ATM operating system; (iii) PCI 3.5.1, reported in summary tables as PCI 3, for rendering stored PAN unreadable where stored PAN and its protection condition are explicitly licensed; (iv) PCI 10a and PCI 10b for the instantiated logging and anomaly-review conditions on the criminal branch; and (v) PCI 9.4.5 electronic-media inventory controls, included as a negative-control check for the NotApplicable branch. Since no electronic media containing CHD is represented as a typed item in any reached situation of this scenario, PCI 9.4.5 remains NotApplicable throughout.

Instantiated check IDs. The executable valuation uses a fixed set of instantiated checks. The patching and anti-malware checks are reported directly under their PCI DSS sub-requirement IDs, PCI DSS 6.3.3 for installation of security patches for `ATM_OS`, and PCI DSS 5.2.1 for anti-malware deployment coverage over `ATM_OS`. Where a principal PCI DSS requirement contains several sub-requirements, the paper uses short labels only for readability. These labels are not additional PCI DSS requirements. In this scenario, `PCI 3` denotes the instantiated stored-account-data unreadability check derived from PCI DSS 3.5.1. `PCI 10a` denotes the instantiated event-logging check derived from PCI DSS 10.2.2. `PCI 10b` denotes the instantiated anomaly or exception-review check derived from PCI DSS 10.4.3. Other PCI 10 sub-requirements, including PCI DSS 10.2.1.1 and PCI DSS 10.4.1.1, may appear in the candidate mapping where their relevance is indicated by the state content, but they are not counted as separate fixed-subset valuation checks in Table 1 unless explicitly instantiated. This preserves a single vocabulary: the candidate mapping table reports underlying PCI DSS sub-requirements, whilst the valuation tables report the fixed instantiated checks used in this experiment.

Instantiated predicate and evidence mapping. The fixed evaluation subset operationalises selected PCI DSS requirement items through explicit model-local applicability and satisfaction predicates. A ViolatedInState outcome requires a licensed basis for applicability along with a licensed condition establishing that the instantiated satisfaction predicate is false. Absence of a satisfaction condition from the reconstructed state is not itself treated as evidence that the control has failed. Operational facts such as malware presence, criminal control, stored-data presence, or connected account context likewise do not independently establish failure unless the corresponding failed control condition is explicitly represented.

The mappings below constitute the operationalisation used for this experiment. They make the evidential basis of each reported valuation explicit, without asserting a unique assessor-independent decomposition of the corresponding PCI DSS requirement item. PCI DSS assessment may depend on implementation context, scope, available evidence, testing procedures, and professional judgement. An assessor-validated implementation may therefore use a different evidential decomposition whilst retaining the same separation between valuation-domain membership, applicability, satisfaction, and state-indexed valuation.

For provenance, the evaluation distinguishes operational facts inherited from the scenario graph or its transition semantics from control-evidence predicates supplied independently for the executable conformance fixture. In particular, `anti_malware_coverage`, `PAN_unreadable`, `required_log_details_present`, and `anomaly_addressed` are not in-

ferred from malware presence, stored-data presence, connected account context, or criminal control. Their Boolean values are supplied explicitly in the model-local evaluation input and are varied independently where a controlled reference case is used. Table 2 records the provenance basis used for the reported states.

Table 1. Scenario 1—state-indexed check-level valuation across reached situations in D_{PCI}. S = SatisfiedInState; V = ViolatedInState; NA = NotApplicable. PCI 3 denotes the instantiated PCI DSS 3.5.1 check. PCI 10 summarises PCI 10a and PCI 10b as defined in the text.

Situation	Threat State	6.3.3	5.2.1	3.5.1	10	9.4.5	Reasoning Summary
S_0	initial vulnerable state	V	S	NA	NA	NA	PCI 6.3.3 is violated, the patch status being F. PCI 5.2.1 is satisfied under the licensed anti-malware deployment condition. Stored PAN, connected logging or review conditions, and electronic media containing CHD remain unlicensed.
S_1	physical compromise	V	V	NA	NA	NA	The physical compromise branch preserves patch-status = F and explicitly licenses anti-malware-coverage = F. PCI 6.3.3 and PCI 5.2.1 are therefore violated under their respective instantiated predicates. Malware presence is represented independently and does not itself determine the PCI 5.2.1 outcome. PCI 3, PCI 10, and PCI 9.4.5 remain not applicable.
S_2	remote compromise	V	V	NA	NA	NA	The remote-compromise branch explicitly licenses patch-status = F and anti-malware-coverage = F, so PCI 6.3.3 and PCI 5.2.1 remain violated. Malware presence is an independent operational condition. The remaining instantiated checks are not applicable.
S_3	adversarial control	V	V	NA	NA	NA	The overtaken state preserves patch-status = F and anti-malware-coverage = F. Malware presence and criminal control are represented as separate operational conditions and do not independently determine either control valuation. Stored PAN and the PCI 10 evidence conditions are still not licensed.
S_4	corrective recovery	S	S	NA	NA	NA	The corrective branch restores patch-status = T and anti-malware-coverage = T, so PCI 6.3.3 and PCI 5.2.1 evaluate as satisfied. The remaining instantiated checks are not applicable.
S_5	card interaction	V	V	NA	NA	NA	The state preserves patch-status = F and anti-malware-coverage = F. Card insertion does not yet license stored PAN or the evidence conditions required by the fixed PCI 10 checks.
S_6	card retention	V	V	NA	NA	NA	The state preserves the failed patching and anti-malware coverage conditions. Stored PAN is not yet licensed for the fixed PCI 3.5.1 check.
S_7	credential exposure	V	V	V	NA	NA	The failed patching and anti-malware coverage conditions persist. PCI 3.5.1 becomes applicable with `stored-PAN` = T, while `PAN-unreadable` = F supplies explicit evidence that the instantiated satisfaction condition fails. PCI 10 remains not applicable.
S_8	cancelled intervention	V	V	V	NA	NA	The cancelled branch preserves patch-status = F, anti-malware-coverage = F, stored-PAN = T, and PAN-unreadable = F. The transaction intervention does not alter these control conditions. The evidence required by the fixed PCI 10 checks is not licensed on this branch.
S_9	connected criminal branch	V	V	V	V	NA	The state preserves the failed patching, anti-malware coverage, and stored-PAN protection conditions. It additionally licenses auditable-event = T with required-log-details-present = F and review-anomaly-identified = T with anomaly-addressed = F. These explicit conditions establish the fixed PCI 10a and PCI 10b violations.
S_{10}	payment to criminals	V	V	V	V	NA	The payment state preserves the explicit conditions supporting the PCI 6.3.3, PCI 5.2.1, PCI 3.5.1, PCI 10a, and PCI 10b violations and carries the same fixed-subset valuation into the repeated-payment loop.

For PCI 5.2.1, the instantiated check concerns anti-malware deployment over the ATM operating-system component. No model-local determination that the represented component is excluded from anti-malware applicability is used in this scenario. Its applicability predicate is

$$\mathrm{App}_{5.2.1,c}(\pi(s)) \equiv \texttt{ATM_OS} \in O_s,$$

and its satisfaction predicate is

$$\mathrm{Sat}_{5.2.1,c}(\pi(s)) \equiv \texttt{anti_malware_coverage} = \mathrm{T}.$$

In addition, malware presence and criminal control do not independently falsify PCI 5.2.1. In the instantiated scenario, `anti_malware_coverage` = T is licensed at the initial state, `anti_malware_coverage` = F is explicitly licensed on the compromised branch, and the corrective transition restores it to T at S_4. The reported PCI 5.2.1 violations

therefore arise from the explicit failed deployment-coverage condition, not from malware presence itself.

For PCI 3.5.1, the instantiated check applies where stored PAN is explicitly licensed:

$$\mathrm{App}_{3.5.1,c}(\pi(s)) \equiv \texttt{stored_PAN} = \mathrm{T},$$

and its satisfaction predicate is

$$\mathrm{Sat}_{3.5.1,c}(\pi(s)) \equiv \texttt{PAN_unreadable} = \mathrm{T}.$$

The presence of `Stored_Debit_Card_Details` identifies the typed data artefact but does not independently determine satisfaction. A PCI 3.5.1 violation is produced only where the state explicitly licenses both `stored_PAN` = T and `PAN_unreadable` = F.

Table 2. Scenario 1—model-local state snapshots, PCI evaluation context, and provenance basis. The provenance column distinguishes operational facts represented by the scenario/transition structure from independently specified control-evidence inputs used by the executable conformance fixture.

Situation	Threat State	O_s (Ontological Items Present)	Key Licensed Conditions in Γ_s Relevant to Instantiated Checks	PCI Evaluation Context	Provenance/Licensing Basis
S_0	initial vulnerable state	{`ATM_OS`}	patch-status = F; update-state = stale; malware-risk = T; anti-malware-coverage = T	True—ATM is represented as payment-channel infrastructure.	Scenario baseline supplies the represented item and operational context. The patch and anti-malware control-status values are explicit model-local evaluation inputs.
S_1	physical compromise	{`ATM_OS`, `Viral_Code`}	patch-status = F; physical-infection = T; malware-present = T; anti-malware-coverage = F	True—compromised ATM operating system is represented as payment-channel infrastructure.	The physical-compromise state/transition supplies the infection facts. The failed anti-malware coverage condition is independently specified and is not inferred from malware presence.
S_2	remote compromise	{`ATM_OS`, `Viral_Code`}	patch-status = F; remote-infection = T; malware-present = T; anti-malware-coverage = F	True—compromised ATM operating system is represented as payment-channel infrastructure.	The remote-compromise state/transition supplies the infection facts. The failed anti-malware coverage condition is independently specified and is not inferred from malware presence.
S_3	adversarial control	{`ATM_OS`, `Viral_Code`}	patch-status = F; malware-present = T; criminal-control = T; anti-malware-coverage = F	True—adversarial control affects payment-channel infrastructure.	The adversarial-control state supplies criminal control and preserves represented compromise facts. The anti-malware control status remains an independently specified control-evidence input.
S_4	corrective recovery	{`ATM_OS`}	patch-status = T; corrective-state = T; anti-malware-coverage = T	True—corrected ATM operating system remains payment-channel infrastructure.	The corrective state supplies the recovery context. Restored patch and anti-malware values are explicit control-evidence inputs associated with this evaluation state and are not inferred merely from malware removal.
S_5	card interaction	{`ATM_OS`, `CardID`}	patch-status = F; card-present = T; criminal-control = T; anti-malware-coverage = F	True—card interaction occurs on represented payment-channel infrastructure.	The card-interaction state supplies the card event, while compromised-branch facts persist. The anti-malware status is supplied independently.
S_6	card retention	{`ATM_OS`, `CardID`}	patch-status = F; card-retained = T; local-data-capture = T; criminal-control = T; anti-malware-coverage = F	True—retained card interaction occurs on represented payment-channel infrastructure.	The card-retention state supplies retention and capture facts; earlier compromise facts persist. The anti-malware status is supplied independently.
S_7	credential exposure	{`ATM_OS`, `Stored_Debit_Card_Details`}	patch-status = F; credentials-exposed = T; criminal-control = T; anti-malware-coverage = F; stored-PAN = T; PAN-unreadable = F	True—stored PAN and its represented protection condition are explicitly licensed in the state snapshot.	The credential-exposure state supplies the typed stored-data artefact and exposure context. The anti-malware and PAN-protection control-status values are independently specified; $\text{PAN-unreadable} = F is - not - inferred - from - stored - PAN = T.$

Table 2. *Cont.*

Situation	Threat State	O_s (Ontological Items Present)	Key Licensed Conditions in Γ_s Relevant to Instantiated Checks	PCI Evaluation Context	Provenance/Licensing Basis
S_8	cancelled intervention	{ATM_OS, Stored_Debit_Card_Details}	patch-status = F; analytics-invoked = T; intervention = T; transaction-executed = F; criminal-control = T; anti-malware-coverage = F; stored-PAN = T; PAN-unreadable = F	True—stored PAN and its represented protection condition persist and intervention occurs before transaction execution.	The intervention state supplies the cancellation/analytics facts. Explicit control-evidence values from the preceding branch are carried forward; no fixed PCI 10 failure condition is supplied on this branch.
S_9	connected criminal branch	{ATM_OS, Stored_Debit_Card_Details, AccountID, CurrentBalance}	patch-status = F; remote-connection = T; account/session-context-licensed = T; payment-executed = F; criminal-control = T; anti-malware-coverage = F; stored-PAN = T; PAN-unreadable = F; auditable-event = T; required-log-details-present = F; review-anomaly-identified = T; anomaly-addressed = F	True—stored PAN and explicit logging and review conditions are licensed on the connected criminal branch.	The connected-branch state supplies account/session and remote-connection context. PAN, anti-malware, logging-detail, and anomaly-response control predicates are explicit evaluation inputs; their failed values are not inferred from connection or criminal control.
S_{10}	payment to criminals	{ATM_OS, Stored_Debit_Card_Details, AccountID, CurrentBalance, AmountExtorted}	patch-status = F; payment-executed = T; transaction-cancelled = F; account/transaction-context-present = T; repeat-payment-loop = T; criminal-control = T; anti-malware-coverage = F; stored-PAN = T; PAN-unreadable = F; auditable-event = T; required-log-details-present = F; review-anomaly-identified = T; anomaly-addressed = F	True—transaction context, stored PAN, and the corresponding anti-malware, logging, and review conditions are explicitly licensed.	The payment state supplies transaction and loop context. The explicit control-evidence values from S_9 are carried into this state; no failed control predicate is inferred from payment execution itself.

PCI 10a instantiates PCI DSS 10.2.2. Its applicability predicate is

$$\mathrm{App}_{10a,c}(\pi(s)) \equiv \texttt{auditable_event} = \mathrm{T},$$

and its satisfaction predicate is

$$\mathrm{Sat}_{10a,c}(\pi(s)) \equiv \texttt{required_log_details_present} = \mathrm{T}.$$

Account or transaction context contributes to the operational setting represented by the state but does not independently establish a logging failure. As for PCI 10a, its violation requires an explicitly represented auditable event together with `required_log_details_present` = F.

PCI 10b instantiates PCI DSS 10.4.3. Its applicability predicate is

$$\mathrm{App}_{10b,c}(\pi(s)) \equiv \texttt{review_anomaly_identified} = \mathrm{T},$$

and its satisfaction predicate is

$$\mathrm{Sat}_{10b,c}(\pi(s)) \equiv \texttt{anomaly_addressed} = \mathrm{T}.$$

Therefore, PCI 10b violation requires an explicitly represented anomaly or exception identified through the instantiated review condition together with `anomaly_addressed` = F.

PCI 6.3.3 and PCI 9.4.5 retain their existing instantiations. PCI 6.3.3 is evaluated from the represented ATM operating-system and patch-status conditions. PCI 9.4.5 remains the negative-control case in this scenario, since no electronic media containing CHD is represented and its applicability predicate consequently remains false.

Model-local state snapshots and PCI-context. Table 2 records the state snapshot $\pi(s) = \langle O_s, \Gamma_s \rangle$ and licensed PCI evaluation context for each reached situation used in the ATM evaluation. These snapshots define the factual basis available to SCF-PCI. Facts absent from O_s, not licensed in Γ_s, or not represented as PCI evaluation context are not used in valuation.

Control-evidence conditions used by the instantiated satisfaction predicates are represented explicitly in Γ_s. A false value reported below is therefore an affirmative model input for the corresponding state and is distinct from the absence of that predicate from the reconstructed snapshot.

Candidate check mapping. Table 3 identifies the PCI DSS checks that are potentially relevant to each reached state on the basis of the typed items, licensed conditions, and PCI evaluation context represented in $\pi(s)$. It defines the bounded space from which instantiated checks are selected for executable valuation. It is not, by itself, a PCI DSS scope determination; formal applicability is resolved by $\mathrm{App}_{r,c}(\pi(s))$ for states in D_{PCI}.

Table 3. Scenario 1—candidate PCI DSS check mapping from model-local state content.

Situation	Threat State	Candidate PCI DSS Checks from Dataset	Model-Local Mapping Rationale
S_0	initial vulnerable state	2.2.1, 5.2.1, 5.3.1, 6.3.1, 6.3.3	The stale ATM operating-system state makes configuration, anti-malware, vulnerability-management, and patching checks potentially relevant.
S_1	physical compromise	5.2.1, 11.5.2, 9.5.1.1, 9.5.1.3	The represented ATM operating system remains subject to the instantiated anti-malware deployment check. The state separately licenses failed anti-malware coverage, while physical infection and malicious code make integrity monitoring, POI inventory, and tampering-awareness checks potentially relevant where the ATM is treated as a POI device.
S_2	remote compromise	5.2.1, 6.3.1, 6.3.3, 11.5.2, 1.4.1, 1.4.2	The represented ATM operating system remains subject to the instantiated anti-malware deployment check and explicitly licenses failed anti-malware coverage. Remote infection also makes vulnerability, patching, and integrity checks potentially relevant. Requirement 1 checks remain conditional on explicit network-boundary facts.
S_3	adversarial control	5.2.1, 7.2.2, 8.2.1, 8.3.1	The represented ATM operating system remains subject to the instantiated anti-malware deployment check, with anti-malware-coverage = F explicitly licensed. Adversarial control separately makes access, identification, and authentication checks potentially relevant.
S_4	corrective recovery	2.2.1, 5.2.1, 6.3.3, 6.5.2	The corrective state restores anti-malware-coverage = T together with the patching condition. Secure configuration and significant-change confirmation remain potentially relevant where reinstall is treated as such in the model.
S_5	card interaction	5.2.1, 9.5.1.1, 9.5.1.3	The represented ATM operating system remains subject to the instantiated anti-malware deployment check with anti-malware-coverage = F. Card interaction also makes POI inventory and tampering-awareness checks potentially relevant.
S_6	card retention	5.2.1, 9.5.1.1, 9.5.1.3, 3.5.1	The represented ATM operating system remains subject to the instantiated anti-malware deployment check with anti-malware-coverage = F. Card retention and local capture make POI checks and conditional stored-account-data unreadability screening potentially relevant. PCI 3.5.1 is not instantiated as applicable until stored PAN and its corresponding protection condition are explicitly licensed.
S_7	credential exposure	5.2.1, 3.5.1, 3.3.1.1, 3.3.1.2, 3.3.1.3	The state explicitly licenses anti-malware-coverage = F for PCI 5.2.1. `Stored_Debit_Card_Details` is also explicitly typed as stored PAN and PAN-unreadable = F supplies the control-evidence condition used by the fixed PCI 3.5.1 instantiation. Stronger PCI 3.3.1.x checks remain conditional on their own explicit artefact and evidence conditions.

Table 3. *Cont.*

Situation	Threat State	Candidate PCI DSS Checks from Dataset	Model-Local Mapping Rationale
S_8	cancelled intervention	5.2.1, 3.5.1, 10.4.1.1, 10.4.3, 10.7.2	Failed anti-malware coverage and the stored-PAN protection conditions persist on the cancelled branch. The intervention state also makes review, exception-handling, and critical-control failure monitoring checks potentially relevant, while the explicit evidence required by the fixed PCI 10a and PCI 10b instantiations is not licensed on this branch.
S_9	connected criminal branch	5.2.1, 3.5.1, 1.4.1, 1.4.2, 10.2.2 (PCI 10a), 10.4.3 (PCI 10b)	The state preserves anti-malware-coverage = F and the stored-PAN protection conditions. The connected branch additionally licenses an auditable event together with the control-evidence conditions used by the fixed logging and anomaly-review checks. PCI 10a is evaluated from the audit-log-detail condition and PCI 10b from the anomaly-identification and response conditions. PCI 8.4.3 remains uninstantiated, as its applicability is confined to authorised remote access by personnel, administrators, or third parties, whereas the present state models an unauthorised criminal connection.
S_{10}	payment to criminals	5.2.1, 3.5.1, 10.2.2 (PCI 10a), 10.4.1.1, 10.4.3 (PCI 10b), 10.2.1.1	The payment state preserves failed anti-malware coverage, the stored-PAN protection conditions, and the explicit auditable-event, audit-log-detail, anomaly-identification, and anomaly-response conditions. In the fixed valuation subset, PCI 10a instantiates 10.2.2 and PCI 10b instantiates 10.4.3; 10.2.1.1 and 10.4.1.1 remain candidate checks unless separately instantiated with their corresponding evidence conditions.

Candidate relevance does not itself establish failed satisfaction. The fixed instantiated checks are valued using the corresponding applicability, satisfaction, and control-evidence conditions documented in Table 4.

Table 4. Instantiated requirement-check predicates and model evidence used in Scenario 1.

Check	Applicability Condition	Satisfaction Condition	Licensed Evidence Required for ViolatedInState
PCI 6.3.3	`ATM_OS` represented	patch-status = T	`ATM_OS` represented and patch-status = F
PCI 5.2.1	`ATM_OS` represented	anti-malware-coverage = T	`ATM_OS` represented and anti-malware-coverage = F
PCI 3.5.1	stored-PAN = T	PAN-unreadable = T	stored-PAN = T and PAN-unreadable = F
PCI 10a/10.2.2	auditable-event = T	required-log-details-present = T	auditable-event = T and required-log-details-present = F
PCI 10b/10.4.3	review-anomaly-identified = T	anomaly-addressed = T	review-anomaly-identified = T and anomaly-addressed = F
PCI 9.4.5	electronic media containing CHD represented	inventory control satisfied for the represented media	applicable electronic media and an explicitly licensed failed inventory condition

4.4. State-Indexed Valuation over the First Detailed Scenario

Table 1 reports the executable check-level valuation for each reached situation in the ATM infection scenario, all of which satisfy $\mathrm{PCIContext}(\pi(s)) = \mathrm{true}$ and therefore belong to D_{PCI}. Each instantiated check receives a single valuation at every evaluated state, applicability is resolved before satisfaction, and valuation changes occur only where the relevant typed item or licensed condition changes in $\pi(s)$. Applicability in this scenario is monotone. Once a check becomes applicable, it remains applicable at every later state, which confines endpoint loss to restoration.

Every **V** entry reported in Table 1 is backed by an explicitly licensed false control-evidence condition identified in Tables 2 and 4. Missing satisfaction evidence is not used as a surrogate false value in these reported cases; operational compromise facts and control-status predicates remain separately represented.

4.5. Path-Sensitive Retention Against a Point-in-Time Endpoint Baseline

The analytical comparator adopts a point-in-time perspective by observing only the final state of an executed scenario path. It provides a deliberately restricted terminal-state projection against which the state-indexed path valuation can be compared. It does not represent conventional PCI DSS assessment practice, which may incorporate historical evidence, documentation, testing, logs, scoping decisions, and assessor judgement. Both views use the same fixed set of instantiated requirement-checks and evaluate only states in D_{PCI}; they differ in whether intermediate states are retained.

The path-level comparison is expressed through the sets V_{path}, V_{end}, and V_{miss} defined in Section 3.4. The endpoint baseline is represented by $V_{\mathrm{end}}(\rho^{\sigma})$, the state-indexed path valuation by $V_{\mathrm{path}}(\rho^{\sigma})$, and endpoint-loss violations by $V_{\mathrm{miss}}(\rho^{\sigma})$. The endpoint-soundness and endpoint-loss results established in Section 3.4.5 provide the formal basis for the comparison reported below.

Table 5 instantiates the path-level construction for the ATM infection scenario. On the criminal and cancelled paths, V_{end} and V_{path} coincide, since the terminal states retain the violated checks encountered along the corresponding execution paths. On the corrective path, V_{miss} is non-empty as S_4 restores both the PCI 6.3.3 patching condition and the PCI 5.2.1 anti-malware deployment condition while the path-level valuation retains their earlier violations. This is an instance of the restoration case characterised in Section 3.4.5. The ATM scenario has monotone applicability and therefore exercises restoration, while the contrasting login-with-spyware scenario exercises applicability-window closure. Across the wider corpus, the same formal construction is instantiated over different combinations of restoration, applicability-window closure, and domain exit.

Table 5. Scenario 1—path-sensitive retention against a point-in-time endpoint baseline. All three reported paths are instantiations of σ over the remote infection branch (E_3/a_{02}) of M, and all reported states belong to D_{PCI}.

Scenario/Path	s_T	$\lvert\rho^{\sigma}\rvert$	$\lvert V_{\mathrm{end}}\rvert$	$\lvert V_{\mathrm{path}}\rvert$	$\lvert V_{\mathrm{miss}}\rvert$	Earliest Violated Checks (t_{first}: check@step)
ATM Infection ρ_{criminal} $(S_0{\to}S_2{\to}S_3{\to}S_5 {\to}S_6{\to}S_7{\to}S_9{\to}S_{10})$	S_{10}	8	5	5	0	PCI 6.3.3@S_0; PCI 5.2.1@S_2; PCI 3.5.1@S_7; PCI 10a@S_9; PCI 10b@S_9
ATM Infection $\rho_{\mathrm{corrective}}$ $(S_0{\to}S_2{\to}S_3{\to}S_4)$	S_4	4	0	2	2	PCI 6.3.3@S_0; PCI 5.2.1@S_2
ATM Infection $\rho_{\mathrm{cancelled}}$ $(S_0{\to}S_2{\to}S_3{\to}S_5 {\to}S_6{\to}S_7{\to}S_8)$	S_8	7	3	3	0	PCI 6.3.3@S_0; PCI 5.2.1@S_2; PCI 3.5.1@S_7

4.6. Cross-Scenario Path-Sensitivity Summary

The ten-scenario corpus [53] is used to instantiate the same executable valuation procedure across different state and path structures. The endpoint-loss taxonomy is defined formally in Section 3.4.5; this subsection reports how its derived cases are realised under the instantiated requirement-checks and model-local PCI evaluation contexts of the evaluated scenarios.

The path-sensitivity labels used here are derived from the truth sequence of the already defined applicability predicate $\mathrm{App}_{r,c}(\pi(s_i))$ along a selected path $\rho^{\sigma} = \langle s_0, \ldots, s_T \rangle$, after applying the PCI evaluation-context gate. They do not represent PCI DSS categories. A check is monotone on a path where, once $\mathrm{App}_{r,c}$ becomes true, it remains true at all later evaluated states. A check is windowed where applicability holds over a bounded interval of the path and later returns to false. A scenario is mixed where its instantiated

checks or branches exhibit more than one such pattern. The labels are therefore descriptive summaries of predicate behaviour over the modelled transition system, computed over the instantiated requirement-checks used for the scenario and only over states inside D_{PCI}.

The ATM scenario realises the restoration mechanism. Its instantiated checks exhibit monotone applicability, so endpoint loss appears only where a previously violated check is restored. The login-with-spyware scenario realises both restoration and applicability-window closure. PCI 8.3.2 is lost by applicability-window closure on every path, whilst the theft cycle at S_{45} produces anti-malware and authentication factor protection violations that are later lost by restoration on paths that traverse it. Domain exit is captured by the formal semantics in Section 3.4.5; where it is exercised in controlled reference inputs, the endpoint falls outside D_{PCI} and therefore receives no check-level valuation.

Table 6 summarises the scenario-level instantiations of the formal endpoint-loss relation after application of the PCI evaluation-context gate. Non-empty V_{miss} cases occur where the corresponding terminal-state projection excludes a violation contained in V_{path}, through restoration, applicability-window closure, or domain exit as characterised in Section 3.4.5. Where none of these mechanisms occurs, V_{end} and V_{path} coincide for the instantiated check set.

Table 6. Cross-scenario summary of path-sensitive behaviour. One scenario (ATM infection) is reported in detail in the main text; a contrasting scenario (login-with-spyware) is summarised through reference cases; all ten are processed through the same valuation pipeline, with check-level valuation applied only to reached states inside D_{PCI}.

Scenario Group	Path-Wise Applicability Pattern	Reporting Level	Mechanism(s) of Endpoint Loss	$\lvert V_{\text{miss}} \rvert$ Range
ATM Infection	Monotone	Detailed in main text	Restoration of previously violated checks before the endpoint state.	0–2
Login-with-Spyware	Windowed	Summarised (reference cases)	Applicability-window closure on every path; restoration on theft paths.	1–3
Remaining evaluated scenarios	Mixed	Reported in artefacts	Restoration, applicability-window closure, both mechanisms, domain exit where the endpoint leaves D_{PCI}, or no endpoint loss depending on the path structure and instantiated checks.	see artefacts

Corpus-level valuation-domain treatment. The corpus comparison in Table 7 reports only finite paths after applying the PCI evaluation-context gate. Where a scenario state has $\text{PCIContext}(\pi(s)) = \text{false}$ or an undetermined PCI evaluation context, the state is retained in the execution trace but excluded from check-level valuation and from the V_{path}, V_{end}, and V_{miss} counts. Thus, the table summarises model-local valuation over states inside D_{PCI}.

Table 7 aggregates the path-level outputs reported in the scenario artefacts. Its purpose is not to introduce additional PCI DSS findings but to show the behaviour of the same valuation procedure across the full ten-scenario corpus. Because each scenario contains multiple finite paths, the table reports ranges for $|V_{\text{path}}|$, $|V_{\text{end}}|$, and $|V_{\text{miss}}|$, including the number of paths on which endpoint loss occurs. This preserves the path-sensitive nature of the result instead of collapsing each scenario to a single endpoint measurement.

Table 7. Scenario corpus—comparison between SCF-PCI state-indexed valuation and an analytical point-in-time endpoint baseline. Ranges are computed over reported finite paths after applying the PCI evaluation-context gate. States with false or undetermined PCI evaluation context are retained in the execution trace but are not assigned check-level valuations. Supplementary loop-invariance rows are excluded from the corpus count.

Scenario	Paths	$\lvert V_{\text{path}}\rvert$ Range	$\lvert V_{\text{end}}\rvert$ Range	$\lvert V_{\text{miss}}\rvert$ Range	Loss Paths	$\sum \lvert V_{\text{miss}}\rvert$
ATM Infection	3	2–5	0–5	0–2	1/3	2
Login-with-Spyware	4	2–4	1	1–3	4/4	8
Money Transfer with Spyware	8	0–5	0–2	0–4	6/8	12
Email Spam Received	5	1–4	0–2	0–3	4/5	9
Sending Email Spam	3	4	0–2	2–4	3/3	10
Update Antivirus with Spyware	6	4–5	1–4	0–4	5/6	15
View Balance with Quid Pro Quo	6	0–4	0–1	0–3	5/6	11
Vishing and Smishing	10	5–6	0–2	4–6	10/10	50
Withdrawal Session Hijacking	4	0–4	0	0–4	1/4	4
View Balance with DDoS	7	0–2	0–2	0–2	3/7	6

The corpus-level outputs instantiate the endpoint-loss taxonomy across the different scenario paths. Under the defined set-difference relation, V_{miss} is empty where the terminal state retains the same violated checks contained in V_{path}, and becomes non-empty where restoration, applicability-window closure, domain exit, or a combination of these mechanisms removes a path-level violation from the terminal-state projection. The reported ranges therefore characterise the behaviour of the implemented valuation procedure over this corpus; they are not an independent empirical measure of the effectiveness of SCF-PCI relative to conventional PCI DSS assessment. The results remain relative to the scenario-specific predicate mappings and the states admitted into D_{PCI}.

4.7. OPA Rego Conformance Checks Against the Valuation Semantics

Finally, we check that the OPA/Rego implementation follows the valuation function defined in Section 3.4.3. These checks confirm that the executable output follows the formal valuation rules used throughout this section; they do not introduce a separate audit or compliance assurance claim. For interactive visualisation of the executable instantiation, SCF-PCI is also provided as a web-based platform through which each scenario can be examined individually, including the reached states, corresponding state snapshots, requirement-check applicability and satisfaction outcomes, state-indexed valuations, and path-level retention results. The platform is available at: https://scfpci.vercel.app (accessed on 8 September 2026).

Selected reference cases are compared with hand-worked expected outputs. The evaluated cases and their expected and observed outcomes are shown in Table 8. They cover all three valuation outcomes inside D_{PCI}: NotApplicable, SatisfiedInState, and ViolatedInState. Controlled changes are then applied to individual model inputs in $\pi(s)$, either by modifying O_s, changing PCI evaluation context, or by adding, removing, or altering a licensed condition in Γ_s. The expected behaviour is that the valuation changes only where the modified input affects the corresponding PCI-context, applicability, or satisfaction predicate.

The controlled reference cases also vary control-evidence predicates while holding the operational compromise facts fixed. This is used to test that an attack fact is not treated as a proxy for control failure. In particular, the paired PCI 5.2.1 cases retain malware presence while changing only anti-malware deployment coverage. Failed satisfaction is asserted only where an explicit false control-evidence value is supplied; absence of the corresponding evidence is not used as the reason for a ViolatedInState result.

Table 8. Reference case conformance checks for state-indexed requirement-check valuation.

Scenario	State	Requirement-Check c	Expected	OPA Output	Reason
ATM Infection	S_0	`PCI_6_3_3_patching`	ViolatedInState	ViolatedInState	$S_0 \in D_{PCI}$; $App_{6.3.3,c}(\pi(S_0))$ holds with `ATM_OS` licensed, whilst $Sat_{6.3.3,c}(\pi(S_0))$ fails under the explicitly licensed patch-status = False condition.
ATM Infection	S_0	`PCI_5_2_1_antimalware`	SatisfiedInState	SatisfiedInState	$S_0 \in D_{PCI}$; $App_{5.2.1,c}(\pi(S_0))$ holds with `ATM_OS` licensed, whilst $Sat_{5.2.1,c}(\pi(S_0))$ holds under anti-malware-coverage = True.
ATM Infection	S_2	`PCI_5_2_1_antimalware`	ViolatedInState	ViolatedInState	$S_2 \in D_{PCI}$; $App_{5.2.1,c}(\pi(S_2))$ holds with `ATM_OS` licensed, while anti-malware-coverage = False explicitly falsifies $Sat_{5.2.1,c}(\pi(S_2))$. Malware presence is represented independently and is not used as the failed deployment condition.
ATM Infection (controlled input)	S_2^+	`PCI_5_2_1_antimalware`	SatisfiedInState	SatisfiedInState	The controlled input retains the S_2 operational compromise facts, including `malware-present`= True, but changes only `anti-malware-coverage` to = True. Applicability therefore remains true while the satisfaction predicate now holds, demonstrating that malware presence and anti-malware deployment coverage are evaluated independently.
ATM Infection	S_0	`PCI_9_4_5_electronic_media_inventory`	NotApplicable	NotApplicable	$S_0 \in D_{PCI}$, but $\neg App_{9.4.5,c}(\pi(S_0))$ because no electronic media containing CHD data is represented as a typed item in the state.
ATM Infection	S_4	`PCI_6_3_3_patching`	SatisfiedInState	SatisfiedInState	$S_4 \in D_{PCI}$; the corrective-state input explicitly supplies patch-status = True, so the patching check remains applicable and is satisfied.
ATM Infection	S_7	`PCI_3_5_1_stored_account_data_unreadability`	ViolatedInState	ViolatedInState	$S_7 \in D_{PCI}$; stored-PAN = True establishes applicability for the fixed PCI 3.5.1 check, while PAN-unreadable = False provides the explicit licensed evidence that its satisfaction condition fails. The presence of `Stored_Debit_Card_Details` alone does not determine the violation.
Login-with-Spyware (controlled reference input)	S_3	`PCI_8_3_2_authentication_factor_protection`	ViolatedInState	ViolatedInState	$S_3 \in D_{PCI}$; $App_{8.3.2,c}(\pi(S_3))$ holds with `Provided_Password` and authentication-factor-in-transmission licensed. The controlled reference input explicitly supplies the failed protection condition, with strong-cryptography = False and unreadability = False; the violation is not inferred from absence of protection evidence.
Login-with-Spyware	S_5	`PCI_8_3_2_authentication_factor_protection`	NotApplicable	NotApplicable	$S_5 \in D_{PCI}$, but authentication factors are no longer in transmission or storage at the successful login terminal, so $\neg App_{8.3.2,c}(\pi(S_5))$. The S_3 reference-case violation is therefore absent from the endpoint through applicability-window closure.

Table 8. *Cont.*

Scenario	State	Requirement-Check c	Expected	OPA Output	Reason
ATM Infection	S_9	`PCI_10a_logging`	ViolatedInState	ViolatedInState	$S_9 \in D_{\text{PCI}}$; auditable-event = True establishes applicability for the instantiated PCI DSS 10.2.2 check, while required-log-details-present = False provides the explicit state evidence for failed satisfaction. Account or session context alone does not determine the violation.
ATM Infection	S_9	`PCI_10b_anomaly_review`	ViolatedInState	ViolatedInState	$S_9 \in D_{\text{PCI}}$; review-anomaly-identified = True establishes applicability for the instantiated PCI DSS 10.4.3 check, while anomaly-addressed = False provides the explicit state evidence for failed satisfaction. Connected account or transaction context does not itself establish failure of the review condition.
ATM Infection (controlled input)	S_7^-	`PCI_3_5_1_stored_account_data_unreadability`	Outside D_{PCI}	Outside D_{PCI}	PCI evaluation context is withdrawn from $\pi(S_7)$, so $S_7^- \notin D_{\text{PCI}}$. The state is retained in the execution trace but receives no check-level valuation, illustrating domain exit as distinct from NotApplicable.

The results in Table 8 confirm that the implementation returns the expected branch of the valuation function inside D_{PCI}, with NotApplicable assigned where PCI evaluation context is licensed but check applicability fails, SatisfiedInState where applicability and satisfaction both hold, and ViolatedInState where applicability holds and an explicitly licensed control-evidence condition establishes failed satisfaction. The paired PCI 5.2.1 cases at S_2 and S_2^+ retain the same malware-present operational fact while changing anti-malware coverage from false to true; the corresponding output changes from ViolatedInState to SatisfiedInState. This counter-case demonstrates that malware presence is not being used as a proxy for anti-malware control failure. The PCI 3.5.1 cases likewise separate stored-data presence from the PAN-protection condition, while the PCI 10 cases separate account or transaction context from the logging and review conditions that determine satisfaction. The login-with-spyware reference case uses an explicitly supplied failed protection condition rather than absence of protection evidence. The same results exercise the endpoint-loss mechanisms established in Section 3.4.5. Restoration is demonstrated when PCI 6.3.3 and PCI 5.2.1 return to SatisfiedInState at S_4 after their earlier violations; applicability-window closure occurs when the PCI 8.3.2 reference case is applicable and violated at S_3 but not applicable at the login terminal; and domain exit occurs when withdrawal of PCI evaluation context places the endpoint outside D_{PCI}, with no check-level valuation assigned. These results confirm that the implementation preserves the distinction between valuation-domain exclusion, check-level non-applicability, operational context, and failed satisfaction under the instantiated predicate mapping.

4.8. Discussion and Interpretive Boundary of the Outputs

A ViolatedInState result is an assessment-support artefact and not a formal PCI DSS finding. Formal determination remains dependent on the wider assessment process, including scope confirmation, evidence sufficiency, source reliability, ROC reporting, and QSA judgement, all of which remain outside the model. This interpretive boundary applies to every result reported in this paper.

The instantiated predicate mappings should therefore be read as explicit model-level operationalisations used to make the evaluation reproducible. They do not assert a unique assessor-independent decomposition of PCI DSS. In assessment practice, implementation

context, scope, available evidence, testing results, and professional judgement may affect how the evidential basis of a requirement item is established. SCF-PCI makes the mapping explicit so that an alternative assessor-validated predicate set can be substituted while preserving the formal distinction between valuation-domain membership, check applicability, satisfaction, and state-indexed valuation.

Within that boundary, the conformance testing results support three observations. First, the implementation preserves the check-level valuation semantics over reached states. Second, path-sensitive valuation retains violations that a terminal-state view omits through restoration, applicability-window closure, or domain exit. Third, $\mathrm{Prof}_r(s)$ preserves the distinct valuations of checks under the same principal requirement, while the optional collapse $C_r(s)$ remains an internal derived summary for downstream analysis.

The interpretation of these observations depends on the structure of contemporary payment and banking infrastructures. Payment security is no longer confined to a single static CDE boundary. Banks, acquirers, payment processors, issuers, merchants, fraud-service providers, cloud platforms, API gateways, authentication services, ATM estates, mobile-payment channels, and monitoring pipelines form an interconnected assurance network. Within such a network, PCI DSS relevance may shift as payment journeys traverse systems and as attacks affect services that support authorisation, transaction processing, logging, authentication, fraud detection, or recovery. A compromised terminal, exposed API path, failed logging pipeline, weakened segmentation rule, or restored authentication service may alter the standards-relevant status of connected components before the final state is reviewed.

The results indicate a modelling consequence for standards-based network and system security analysis. PCI DSS valuation over networked payment systems cannot always be reduced to a terminal-state judgement without loss of information. A requirement-check may become applicable only during a bounded attack interval, a control condition may fail and later recover, or a state may leave the PCI valuation domain before endpoint inspection. These cases are not anomalies of implementation; they follow from the interaction between dynamic operational states and standards-derived applicability conditions. The path-sensitive layer makes that interaction explicit by retaining the valuation history of each instantiated requirement-check across the reached states.

The interpretation also reframes the role of evidence. SCF-PCI does not infer unrepresented organisational facts, live telemetry, assessor judgement, or formal PCI DSS scope determinations. It evaluates only the typed items, licensed conditions, events, actions, and PCI evaluation context represented in the model. In an operational setting, the assurance network would need to connect state representations to reliable evidence sources such as transaction logs, authentication records, SIEM alerts, EDR signals, configuration records, segmentation evidence, API access logs, incident timelines, fraud-case notes, and control-status records. Without such evidence, a state-indexed valuation remains a model-local result.

The broader implication is that standards assurance for next-generation payment infrastructures requires an explicit account of state, path, and evidential support. SCF-PCI contributes that account for PCI DSS requirement-check valuation, whilst preserving the distinction between model-level standards analysis and formal PCI DSS assessment. The conditions under which operational state representations can be produced are taken up in Appendix A.

5. Limitations and Future Work

The predicates used in this study provide one systematic and publicly grounded formalisation of selected PCI DSS requirement items, yet they are not authoritative PCI

DSS interpretations. The framing of the work was informed by practitioner engagement with personnel from the banking sector at a UK O-SII, whilst the predicate mapping, requirement-check decomposition, and state-level valuations have not yet been validated against QSA workpapers, assessment files, ROC determinations, or formal scoping decisions. Different assessment teams may legitimately decompose the same PCI DSS requirement into different observable conditions, or treat the same asset, service, control condition, scope relationship, and compensating evidence differently within assessment practice. This affects how SCF-PCI outputs should be interpreted, since identical system evidence may lead to different applicability and satisfaction outcomes under a different assessor-validated predicate mapping.

The same issue applies to PCI DSS scope. SCF-PCI admits a state into the valuation domain only where a licensed PCI evaluation context is represented, creating a model-local gate that does not constitute a PCI DSS scoping decision. In connected banking and payment environments, scope may depend on shared services, segmentation evidence, service-provider relationships, cloud-hosted components, API gateways, identity services, monitoring platforms, backup systems, or systems that can affect the security of the CDE. If these relationships are absent from the state representation, SCF-PCI may fail to value a state that would require assessor attention; if they are represented too broadly, the model may attach PCI relevance to states that a formal assessment would treat differently. The framework makes the valuation basis explicit, but does not resolve the underlying scoping judgement.

SCF-PCI also assumes an ordered state sequence in which typed items and licensed conditions are available for valuation, whilst real assurance work often depends on incomplete, uneven, and time-distributed evidence. Logs may be missing, clocks may not align, cloud resources may appear and disappear, API traces may span several platforms, and incident timelines may combine SIEM alerts, EDR events, fraud-case notes, authentication records, and manual investigation findings. Posture facts such as patch status, anti-malware coverage, encryption configuration, logging status, or segmentation rules may be derived from configuration systems and endpoint tooling, whereas conditions such as criminal control, credential exposure, malware execution, CHD exposure, or unauthorised API use are harder to observe during an incident and may only be confirmed through subsequent forensic analysis [56]. The current framework uses crisp predicates, so a condition is either licensed for the state or excluded from valuation. Practical use would require provenance, timestamp quality, source reliability, evidential confidence, and subsequent revision of the incident record to be associated with the predicates over $\pi(s)$.

Modern payment architecture also raises asset-identity problems that the current framework only partially addresses. PCI DSS valuation depends on the relevant system component, its persistence across states, and its relationship to the CDE. These assumptions are harder to maintain where payment services rely on cloud workloads, containers, serverless functions, API-mediated processing, tokenisation services, and third-party platforms. A component may be short-lived, replicated, renamed, replaced, or visible only through logs and configuration records. SCF-PCI currently assumes that items can be represented consistently across the scenario path, so weak asset identity in operational environments may affect applicability, satisfaction, and endpoint-loss results.

The evaluation remains scenario-bound at this stage. The reported V_{miss} results demonstrate that endpoint-only valuation can lose intermediate violations under the specified semantics and scenario paths, but they do not estimate how frequently such loss occurs in operational PCI DSS assessments or across banking estates. Broader validation would require a larger corpus of payment-security incidents, tabletop scenarios, forensic reconstructions, and assessor-reviewed cases. The present evaluation supports semantic and

executable conformance, not operational prevalence or infrastructure-scale performance. For substantially larger payment infrastructures, the evaluation workload is expected to depend on the number of represented states, instantiated requirement-checks, and admissible execution paths. Branching and cyclic transition structures may expand the path space that must be considered, particularly where multiple alternative operational trajectories are retained for analysis. Scalability at this level has not yet been empirically assessed and remains outside the scope of the present formalisation.

SCF-PCI is most suitable where a discretised state sequence already exists, as in post-incident forensic analysis, where investigators may reconstruct an ordered incident narrative [57], or in pre-incident threat modelling and tabletop exercises, where states can be authored deliberately. Continuous assurance workflows would require reliable state sequences to be reconstructed from heterogeneous operational sources such as transaction logs, authentication records, SIEM alerts, EDR signals, configuration records, segmentation evidence, API access logs, fraud-case notes, incident timelines, and control-status records. Automatic construction of such sequences remains outside the present evaluation.

Future work should prioritise external validation of the predicate mappings with QSAs, PCI assurance teams, payment-security practitioners, and cyber-risk teams, followed by provenance-aware and uncertainty-aware valuation, stronger asset identity handling across ephemeral cloud components and third-party service dependencies, and infrastructure-scale evaluation using state abstraction, path pruning, bounded evaluation, and prioritisation of control-critical branches. Machine-checked verification of the valuation core remains useful [58], but assessment relevance will depend on evidence quality, predicate validation, and integration with real assurance workflows.

6. Conclusions

SCF-PCI formalises requirement-check valuation over adversarially evolving payment-system states by separating PCI evaluation-context membership, requirement applicability, and control satisfaction at each reached state. Across a single execution path, a requirement-check may move in and out of applicability and between satisfaction and violation as compromise, intervention, and remediation unfold. The path-level construction retains these state-indexed valuations and defines V_{miss} as the set of violations present along the evaluated path but absent from its terminal-state projection. Restoration, applicability-window closure, and domain exit characterise the three endpoint conditions under which this derived relation becomes non-empty.

The executable conformance evaluation shows that these semantics can be realised as a trace-level procedure. PCI evaluation context is resolved before valuation, applicability before satisfaction, and check-level outcomes are assembled into requirement profiles across the reached states. The evaluated scenarios instantiate the formal construction over different adversarial state and path structures and exercise the endpoint-loss mechanisms defined by the model. The point-in-time endpoint baseline used in this comparison is an analytical terminal-state projection and is not intended to represent conventional PCI DSS assessment practice.

For assurance workflows in banking and payment security, the formalism provides a basis for evaluating standards-derived predicates over evolving operational states. Questions such as when a control condition failed during an intrusion, whether a payment component became PCI-relevant during an attack path, and whether remediation restored the relevant condition can be represented directly through the state-indexed valuation history.

Translation of these semantics into a working assurance capability depends on their connection to the evidential conditions of live payment environments. Future work should examine how state-indexed reasoning can be grounded in operational evidence across the

assurance network, including transaction logs, authentication records, API traces, SIEM alerts, EDR signals, segmentation evidence, fraud-case notes, and incident timelines, whilst preserving the distinction between model-local requirement-check valuation and formal PCI DSS assessment. SCF-PCI therefore provides a formal basis for state-aware standards assurance in which requirement-check applicability and satisfaction can be evaluated across the progression of an operational payment system.

Author Contributions: Conceptualisation, F.Y.L. and K.O.; methodology, F.Y.L., K.O. and A.P.; formal analysis, F.Y.L., K.O. and A.P.; software, F.Y.L.; validation, F.Y.L., K.O. and A.P.; investigation, F.Y.L.; resources, F.Y.L., K.O. and A.P.; writing—original draft preparation, F.Y.L.; writing—review and editing, F.Y.L., K.O. and A.P.; visualisation, F.Y.L.; supervision, K.O. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The validated scenarios used in this paper are implemented in a web-based platform that exposes the accessible portion of the full ontology. The platform embeds SCF-PCI as an executable framework for the interactive exploration of state-indexed compliance outputs and is available at https://scfpci.vercel.app (accessed on 8 September 2026). The corresponding workbook dataset, which contains the ten scenarios and instructions on how they should be read, is available at [53]. All analyses and examples presented in this paper remain within the permitted research scope and do not disclose proprietary or confidential information.

Conflicts of Interest: Author Anthony Phipps was employed by the company "Lloyds Banking Group (United Kingdom), London, United Kingdom". The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Appendix A. Executable Realisation and Typed Artefacts

Appendix A gives the executable counterpart of the formal construction in Section 3. It records how the world model, dynamic rules, transition system, bounded execution paths, state snapshots, PCI evaluation context, and requirement-check valuations are materialised as typed artefacts. The formal semantics are defined in the main text and the purpose here is implementation traceability.

Appendix A.1. Typed Artefact Inventory

Table A1 summarises the artefacts used by the executable instantiation. They correspond to the four levels of the framework and provide the typed interface through which scenario states, transitions, snapshots, PCI-context gates, and valuations are represented.

Appendix A.2. Ontological Substrate

The executable realisation imports the world model

$$\Sigma = (\mathcal{T}Box, RBox, \mathcal{A}Box)$$

as a fixed semantic substrate. The *TBox* and *RBox* provide the vocabulary, typing constraints, and admissible relations used by the executable artefacts. Individuals asserted in the *ABox* are materialised as typed references, including `SituationRef`, `ItemRef`, `ConditionRef`, `EventRef`, and `ThreatRef`. These references provide the typed input for rule execution, path construction, state-snapshot materialisation, PCI-context evaluation, and state-indexed valuation.

Appendix A.3. Dynamic Rule Execution and Transition Induction

Each rule in R_{dyn} is evaluated over a typed `DynamicRuleContext` containing the current situation and the events asserted for the scenario. A triggered rule produces a `DynamicRuleResult`, recording the induced action, successor situation, and originating SWRL clause. The resulting action assertions are collected as transitions in $\rightarrow$, yielding the executable transition system

$$\mathcal{M} = (S, A, \rightarrow).$$

This implements the behavioural level from Section 3 without adding normative content.

Appendix A.4. Bounded Path Selection

The transition system may contain branching and cyclic behaviour. Execution is therefore parameterised by a path selector

$$\sigma : \mathcal{P}_+(S \times A \times S) \rightarrow (S \times A \times S),$$

where $\mathcal{P}_+$ denotes the set of non-empty subsets. Given the set $\Delta(s)$ of admissible transitions from a reached situation s, the selector σ chooses one transition and induces a selected execution path ρ^σ.

The selected path is a finite execution prefix. If $\mathcal{M}$ contains a cycle or self-transition, the transition system may admit infinite runs, but executable evaluation is bounded by a maximum depth $N_{\max}$. Different selectors over the same $\mathcal{M}$ can realise the criminal fraud path, corrective remediation path, or analytic intervention path, whilst the full branching structure remains available independently of any selected run.

Table A1. Typed artefacts across the four SCF-PCI levels.

Construct	Formal Object	Typed Object	Fields	Implementation Role in SCF-PCI
Level 1. Ontological substrate over world model Σ				
Situation	$s \in S$	`SituationRef`	`id, label`	Typed reference for a reached or reachable operational state. It is used as a state node for transition construction, path construction, snapshot materialisation, and state-indexed valuation.
Item	O_s	`ItemRef`	`id, type`	Typed reference for an operational object represented in a situation and available for snapshot materialisation.
Condition	Γ_s	`ConditionRef`	`id, type, holds_in, about`	Typed reference for a security-relevant property attached to an item in a situation. It supplies state-local input to applicability and satisfaction predicates.
Event	`occur_in`	`EventRef`	`id, label, occur_in`	Typed reference for an occurrence asserted in a scenario. Events provide inputs to dynamic rule execution and provenance for reached states.
Threat	–	`ThreatRef`	`id, label`	Typed reference for adversarial context represented in the scenario.
Level 2. Dynamic rule execution and transition induction over R_{dyn}				
Dynamic rule	$d \in R_{\text{dyn}}$	`DynamicRule`	`id, evaluate(ctx)`	Executable behavioural or threat rule evaluated over a typed rule context.
Rule context	Rule body	`DynamicRule Context`	`situation, events`	Typed input passed to a dynamic rule during execution.
Rule result	Rule head	`DynamicRule Result`	`triggered, action, to, trace`	Result object recording whether the rule fired and, where it fired, the successor situation, induced action, and originating rule trace.
Transition edge	$(s, a, s') \in \rightarrow$	`ActionRef`	`id, from, to`	Labelled transition induced by dynamic rule execution.
Transition system	$\mathcal{M} = (S, A, \rightarrow)$	`Transition SystemRef`	`states, actions, transitions`	Executable container for the induced scenario transition structure.

Table A1. *Cont.*

Construct	Formal Object	Typed Object	Fields	Implementation Role in SCF-PCI
Level 3. Bounded path selection, snapshot materialisation, and PCI-context gating				
Path selector	σ	`PathSelector`	`select` $(\Delta(s))$	Selection function used to choose one admissible transition from the outgoing transition set of a reached situation.
Execution path	ρ^{σ}	`Execution PathRef`	`states, max_depth`	Typed record of the selected finite execution prefix, bounded by $N_{\max}$.
State snapshot	$\pi(s) = \langle O_s, \Gamma_s \rangle$	`State SnapshotRef`	`situation, items, conditions, provenance`	Materialised state object passed to PCI-context, applicability, and satisfaction predicates.
PCI-context	PCIContext $(\pi(s))$	`PCIContext Ref`	`situation, context, basis, source`	Typed record of whether the reached state has a licensed PCI evaluation context.
Trace step	$s_i \in \rho^{\sigma}$	`ComplianceStep`	`situation, snapshot, pci_context, findings`	Typed carrier for one reached situation, its snapshot, its PCI-context status, and any attached requirement-check findings.
Level 4. Normative trace preparation and semantic valuation				
Domain map	$s \in D_{\text{PCI}}$	`PCIValuation DomainMap`	`bindings`	Implementation record of how each reached situation is treated by the PCI valuation gate.
Applicability map	$\text{App}_{r,c}$ $(\pi(s))$	`ApplicableCheck StateMap`	`bindings`	Implementation record of the reached situations in which instantiated checks are applicable.
Check finding	$(r, c,$ $\pi(s))$	`Requirement CheckFinding`	`rule, check, snapshot, app, sat, valuation`	Typed record binding one instantiated requirement-check to one reached state snapshot and its resulting valuation.
Valuation	$V_{r,c}(s)$	`ValuationRef`	`outcome`	Typed carrier for one check-level valuation outcome.
Requirement profile	$\text{Prof}_r(s)$	`Requirement ProfileRef`	`rule, valuations`	Typed grouping of all check-level valuations under one principal-requirement at one reached situation.
Derived summary	$C_r(s)$	`CollapseRef`	`rule, profile, summary`	Optional implementation object used only where a downstream analysis requires one derived requirement-level status.

Appendix A.5. State Snapshot and PCI-Context Materialisation

For each reached situation $s_i \in \rho^{\sigma}$, the executable realisation materialises the snapshot

$$\pi(s_i) = \langle O_{s_i}, \Gamma_{s_i} \rangle$$

as a typed $\texttt{StateSnapshotRef}_{s_i}$. The item component O_{s_i} is obtained from the typed situation content, while the condition component Γ_{s_i} is derived from licensed scenario content, transition semantics, and explicit model-local evaluation inputs with recorded provenance.

The same snapshot is used to materialise a `PCIContextRef`. This record stores the PCI evaluation context for the reached situation and is attached to the corresponding trace step before requirement-check valuation is attempted.

Appendix A.6. Requirement-State Binding and Valuation Trace

For each reached situation $s_i \in \rho^{\sigma}$, the compliance trace stores a `ComplianceStep` containing the corresponding `StateSnapshotRef`, `PCIContextRef`, and list of findings. Requirement-check findings are created only for steps whose PCI evaluation context permits valuation. For each such step, each principal-requirement $r \in \text{Req}$ and each instantiated check $c \in \text{Checks}(r)$ admitted to check-level valuation after evidence resolution gives rise to a `RequirementCheckFinding`; unresolved applicable pairs are retained separately in `UnresolvedEvidenceMap`. This record stores the requirement ID, check ID, snapshot reference, applicability result, satisfaction result where applicable, and eventual valuation.

Algorithm A2 evaluates $\text{App}_{r,c}(\pi(s))$ and $\text{Sat}_{r,c}(\pi(s))$ only where the PCI-context gate has been satisfied. Where the required satisfaction evidence is unresolved, the pair is retained in `UnresolvedEvidenceMap`; otherwise, the resulting value is stored as a typed

`ValuationRef`. The check-level values are then assembled into $\mathrm{Prof}_r(s)$, with $C_r(s)$ stored only where a derived single requirement-level status is needed.

Appendix A.7. Execution and Valuation Algorithms

Algorithm A1 materialises the execution trace from the world model, dynamic rules, initial situation, scenario stimulus, path selector, and depth bound. Algorithm A2 attaches requirement-check valuations to the resulting trace.

Algorithm A1 Execution and State Snapshot Materialisation.

Require: World model Σ; dynamic rules R_{dyn}; initial situation s_0; scenario stimulus; path selector σ; maximum depth $N_{\max}$
Ensure: Transition system $\mathcal{M}$; selected execution prefix ρ^{σ}; typed trace $\mathfrak{T}$

Materialise Σ as typed ontology references: `SituationRef`, `ItemRef`, `ConditionRef`, `EventRef`, and `ThreatRef`
Extract S and, for each $s \in S$, derive O_s and licensed Γ_s
Initialise $\rightarrow \leftarrow \emptyset$
for each $s \in S$ **do**
    Construct `DynamicRuleContext`$(s, \mathrm{events}(s))$
    for each $d \in R_{\mathrm{dyn}}$ **do**
        Evaluate d over the context
        if d triggers successor s' through action a **then**
            Append typed transition (s, a, s') to $\rightarrow$
        end if
    end for
end for
Define $\mathcal{M} \leftarrow (S, A, \rightarrow)$
Select a bounded execution prefix ρ^{σ} from $\mathcal{M}$ using σ and $N_{\max}$
Initialise $\mathfrak{T} \leftarrow [\,]$
for each reached situation $s_i \in \rho^{\sigma}$ **do**
    Materialise $\pi(s_i) = \langle O_{s_i}, \Gamma_{s_i} \rangle$ as `StateSnapshotRef`
    Evaluate PCIContext$(\pi(s_i))$ and materialise the result as `PCIContextRef`
    Append `ComplianceStep`$(s_i, \pi(s_i), \mathtt{PCIContextRef}, [\,])$ to $\mathfrak{T}$
end for
return $\mathcal{M}$, ρ^{σ}, $\mathfrak{T}$

The four formal levels are therefore realised as typed executable artefacts. The main-text evaluation uses bounded executions over the common transition system and applies requirement-check valuation only to reached situations with licensed PCI evaluation context.

Algorithm A2 Normative Trace Preparation and Requirement-Check Valuation.

Require: Typed trace $\mathfrak{T}$; requirement set Req; instantiated checks Checks(r)
Ensure: Valuation-annotated trace $\mathfrak{T}^{\star}$; `ApplicableCheckStateMap`; `PCIValuationDomainMap`; `UnresolvedEvidenceMap`

Initialise $\mathfrak{T}^{\star} \leftarrow \mathfrak{T}$
Initialise `ApplicableCheckStateMap` $\leftarrow [\,]$
Initialise `PCIValuationDomainMap` $\leftarrow [\,]$
Initialise `UnresolvedEvidenceMap` $\leftarrow [\,]$
for each `ComplianceStep` $t_i \in \mathfrak{T}^{\star}$ **do**
    Let $s \leftarrow t_i$.`situation`, $\pi(s) \leftarrow t_i$.`snapshot`, and $p \leftarrow t_i$.`PCIContextRef`
    Initialise findings $F_i \leftarrow [\,]$
    if $p \neq$ true **then**
        Append $(s, p,$ OutsideValuationDomain$)$ to `PCIValuationDomainMap`
        Attach F_i to t_i
        continue
    end if

Algorithm A2 *Cont.*

Append $(s, p, \text{InValuationDomain})$ to `PCIValuationDomainMap`
for each $r \in \text{Req}$ and each $c \in \text{Checks}(r)$ **do**
  if $\neg \text{App}_{r,c}(\pi(s))$ **then**
    $v \leftarrow \text{NotApplicable}$
  else
    Append (r, c, s) to `ApplicableCheckStateMap`
    if the licensed evidence required to resolve $\text{Sat}_{r,c}(\pi(s))$ is incomplete **then**
      Append (r, c, s) to `UnresolvedEvidenceMap`
      ▷ missing evidence is not treated as control failure
      continue
    end if
    if $\text{Sat}_{r,c}(\pi(s))$ **then**
      $v \leftarrow \text{SatisfiedInState}$
    else
      $v \leftarrow \text{ViolatedInState}$
    end if
  end if
  Append `RequirementCheckFinding`$(r, c, \pi(s), v)$ to F_i
end for
Attach F_i to t_i
Assemble $\text{Prof}_r(s)$ for each represented requirement r
Optionally store $C_r(s)$ only where a derived single requirement-level status is needed
end for
return $\mathfrak{T}^\star$, `ApplicableCheckStateMap`, `PCIValuationDomainMap`, `UnresolvedEvidenceMap`

References

1. Cherif, A.; Badhib, A.; Ammar, H.; Alshehri, S.; Kalkatawi, M.; Imine, A. Credit card fraud detection in the era of disruptive technologies: A systematic review. *J. King Saud Univ.-Comput. Inf. Sci.* **2023**, *35*, 145–174. [CrossRef]
2. PCI Security Standards Council. *Payment Card Industry Data Security Standard (PCI DSS)*, version 4.0.1; PCI Security Standards Council: Wakefield, MA, USA, 2024.
3. Chippagiri, S.; Ramesh, A. PCI DSS: A Critical Analysis of Implementation, Effectiveness, and Legislative Impact in Payment Card Security. *Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol.* **2025**, *11*, 1258–1266. [CrossRef]
4. PCI Security Standards Council. *PCI DSS v4.0.1 Report on Compliance (ROC) Template*; Revision 3; PCI Security Standards Council: Wakefield, MA, USA, 2024.
5. Loumachi, F.Y.; Lacerda, M.J.; Ouazzane, K.; Adnane, A.; Adamyk, O. AI in control: Rethinking cybersecurity compliance and auditing. *Inf. Softw. Technol.* **2026**, *195*, 108132. [CrossRef]
6. Lunghi, D.; Simitsis, A.; Caelen, O.; Bontempi, G. Adversarial Learning in Real-World Fraud Detection: Challenges and Perspectives. In *DEC '23: Proceedings of the Second ACM Data Economy Workshop*; Association for Computing Machinery (ACM): New York, NY, USA, 2023; pp. 27–33.
7. Dal Pozzolo, A.; Boracchi, G.; Caelen, O.; Alippi, C.; Bontempi, G. Credit Card Fraud Detection: A Realistic Modeling and a Novel Learning Strategy. *IEEE Trans. Neural Networks Learn. Syst.* **2018**, *29*, 3784–3797. [CrossRef]
8. Sowinski-Mydlarz, V.; Vassilev, V.; Ouazzane, K.; Phipps, A. Security Analytics Framework Validation Based on Threat Intelligence. In *Proceedings of the 2022 International Conference on Computational Science and Computational Intelligence (CSCI)*; IEEE: New York, NY, USA, 2022; pp. 942–947.
9. Carminati, M.; Polino, M.; Continella, A.; Lanzi, A.; Maggi, F.; Zanero, S. Security Evaluation of a Banking Fraud Analysis System. *ACM Trans. Priv. Secur.* **2018**, *21*, 11. [CrossRef]
10. Seaman, J. *PCI DSS*, 1st ed.; Apress: Berkeley, CA, USA, 2020.
11. Bataityte, K.; Vassilev, V.; Gill, O.J. Ontological Foundations of Modelling Security Policies for Logical Analytics. In *Proceedings of the Artificial Intelligence Applications and Innovations*; IFIP Advances in Information and Communication Technology; Springer International Publishing: Cham, Switzerland, 2020; Volume 583, pp. 368–380.
12. Bataityte, K.; Vassilev, V.; Gill, O.J. Finding Logical Vulnerability in Policies Using Three-Level Semantic Framework. In *Proceedings of the Artificial Intelligence Applications and Innovations*; Maglogiannis, I., Iliadis, L., Macintyre, J., Avlonitis, M., Papaleonidas, A., Eds.; Springer International Publishing: Cham, Switzerland, 2024; pp. 276–289.

13. Snyder, H. Literature Review as a Research Methodology: An Overview and Guidelines. *J. Bus. Res.* **2019**, *104*, 333–339. [CrossRef]
14. Webster, J.; Watson, R.T. Analyzing the Past to Prepare for the Future: Writing a Literature Review. *MIS Q.* **2002**, *26*, xiii–xxiii. [CrossRef]
15. Baier, C.; Katoen, J.P. *Principles of Model Checking*; MIT Press: Cambridge, MA, USA, 2008.
16. Zeigler, B.P.; Muzy, A.; Kofman, E. *Theory of Modeling and Simulation: Discrete Event and Iterative System Computational Foundations*, 3rd ed.; Academic Press: Cambridge, MA, USA, 2018.
17. Harel, D. Statecharts: A Visual Formalism for Complex Systems. *Sci. Comput. Program.* **1987**, *8*, 231–274. [CrossRef]
18. Morse, E.A.; Raval, V. PCI DSS: Payment card industry data security standards in context. *Comput. Law Secur. Rep.* **2008**, *24*, 540–554. [CrossRef]
19. PCI Security Standards Council. *Information Supplement: Guidance for PCI DSS Scoping and Network Segmentation*; PCI Security Standards Council: Wakefield, MA, USA, 2016.
20. PCI Security Standards Council. *PCI DSS Quick Reference Guide: Understanding the Payment Card Industry Data Security Standard Version 3.1*; For Merchants and Other Entities Involved in Payment Card Processing; PCI Security Standards Council: Wakefield, MA, USA, 2015.
21. National Institute of Standards and Technology. *OSCAL Assessment Layer: Assessment Results Model*; NIST OSCAL: Gaithersburg, MD, USA, 2025 .
22. National Institute of Standards and Technology. *Open Security Controls Assessment Language (OSCAL)*; Computer Security Resource Center (CSRC): Gaithersburg, MD, USA, 2026.
23. OASIS LegalRuleML Technical Committee. *LegalRuleML Version 1.0*; OASIS Standard: Woburn, MA, USA, 2021.
24. Governatori, G. The Regorous Approach to Process Compliance. In *2015 IEEE 19th International Enterprise Distributed Object Computing Workshop*; IEEE: New York, NY, USA, 2015, pp. 33–40.
25. Cejas, O.A.; Azeem, M.I.; Abualhaija, S.; Briand, L.C. NLP-Based Automated Compliance Checking of Data Processing Agreements Against GDPR. *IEEE Trans. Softw. Eng.* **2023**, *49*, 4282–4303. [CrossRef]
26. Zhong, H.; Xiao, C.; Tu, C.; Zhang, T.; Liu, Z.; Sun, M. How Does NLP Benefit Legal System: A Summary of Legal Artificial Intelligence. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Online, 5–10 July 2020*; Jurafsky, D., Chai, J., Schluter, N., Tetreault, J., Eds.; Association for Computational Linguistics: Stroudsburg, PA, USA, 2020; pp. 5218–5230.
27. Chalkidis, I.; Jana, A.; Hartung, D.; Bommarito, M.; Androutsopoulos, I.; Katz, D.; Aletras, N. LexGLUE: A Benchmark Dataset for Legal Language Understanding in English. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Dublin, Ireland, 22–27 May 2022*; Muresan, S., Nakov, P., Villavicencio, A., Eds.; Association for Computational Linguistics: Stroudsburg, PA, USA, 2022; pp. 4310–4330.
28. Leucker, M.; Schallhart, C. A Brief Account of Runtime Verification. *J. Log. Algebr. Program.* **2009**, *78*, 293–303. [CrossRef]
29. Bauer, A.; Leucker, M.; Schallhart, C. Runtime Verification for LTL and TLTL. *ACM Trans. Softw. Eng. Methodol.* **2011**, *20*, 14. [CrossRef]
30. Dinesh, N.; Joshi, A.; Lee, I.; Sokolsky, O. Checking Traces for Regulatory Conformance. In *Proceedings of the Runtime Verification*; Leucker, M., Ed.; Springer: Berlin/Heidelberg, Germany, 2008; pp. 86–103.
31. DeYoung, H.; Garg, D.; Jia, L.; Kaynar, D.; Datta, A. Experiences in the logical specification of the HIPAA and GLBA privacy laws. In *WPES '10: Proceedings of the 9th Annual ACM Workshop on Privacy in the Electronic Society*; Association for Computing Machinery (ACM): New York, NY, USA, 2010; pp. 73–82.
32. Basin, D.; Klaedtke, F.; Müller, S.; Zălinescu, E. Monitoring Metric First-Order Temporal Properties. *J. ACM* **2015**, *62*, 15. [CrossRef]
33. Basin, D.; Harvan, M.; Klaedtke, F.; Zălinescu, E. MONPOLY: Monitoring Usage-Control Policies. In *Proceedings of the Runtime Verification*; Khurshid, S., Sen, K., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; pp. 360–364.
34. Chowdhury, O.; Jia, L.; Garg, D.; Datta, A. Temporal Mode-Checking for Runtime Monitoring of Privacy Policies. In *Proceedings of the Computer Aided Verification*; Biere, A., Bloem, R., Eds.; Springer International Publishing: Cham, Switzerland, 2014; pp. 131–149.
35. van Beest, N.; Groefsema, H.; Cryer, A.; Governatori, G.; Colombo Tosatto, S.; Burke, H. Cross-Instance Regulatory Compliance Checking of Business Process Event Logs. *IEEE Trans. Softw. Eng.* **2023**, *49*, 4917–4931. [CrossRef]
36. Rinderle-Ma, S.; Winter, K.; Benzin, J.V. Predictive Compliance Monitoring in Process-Aware Information Systems: State of the Art, Functionalities, Research Directions. *Inf. Syst.* **2023**, *115*, 102210. [CrossRef]
37. Klessascheck, F.; Pufahl, L. Reviewing Uses of Regulatory Compliance Monitoring. *Softw. Syst. Model.* **2025**, *25*, 413–436. [CrossRef]
38. Pesic, M.; Van Der Aalst, W.M.P. A Declarative Approach for Flexible Business Processes Management. In *Proceedings of the Business Process Management Workshops*; Eder, J., Dustdar, S., Eds.; Springer: Berlin/Heidelberg, Germany, 2006; pp. 169–180.

39. Maggi, F.M.; Montali, M.; Westergaard, M.; Van Der Aalst, W.M.P. Monitoring Business Constraints with Linear Temporal Logic: An Approach Based on Colored Automata. In *Business Process Management (BPM 2011)*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 132–147.
40. van der Aalst, W.M.P. *Process Mining: Data Science in Action*, 2nd ed.; Springer: Berlin/Heidelberg, Germany, 2016.
41. Hashmi, M.; Governatori, G.; Lam, H.P.; Wynn, M.T. Are We Done with Business Process Compliance: State of the Art and Challenges Ahead. *Knowl. Inf. Syst.* **2018**, *57*, 79–133. [CrossRef]
42. Vasarhelyi, M.A.; Alles, M.G.; Williams, K.T. *Continuous Assurance for the Now Economy. A Thought Leadership Paper for the Institute of Chartered Accountants in Australia*; Institute of Chartered Accountants in Australia: Sydney, Australia, **2010**.
43. Strelcenia, E.; Prakoonwit, S. Generating Synthetic Data for Credit Card Fraud Detection Using GANs. In *Proceedings of the 2022 International Conference on Computers and Artificial Intelligence Technologies (CAIT)*; IEEE: New York, NY, USA, 2022; pp. 42–47.
44. Xu, L.; Skoularidou, M.; Cuesta-Infante, A.; Veeramachaneni, K. Modeling tabular data using conditional GAN. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*; Curran Associates Inc.: Red Hook, NY, USA, 2019.
45. Chatterjee, P.; Das, D.; Rawat, D.B. Digital twin for credit card fraud detection: Opportunities, challenges, and fraud detection advancements. *Future Gener. Comput. Syst.* **2024**, *158*, 410–426. [CrossRef]
46. Sheyner, O.; Haines, J.; Jha, S.; Lippmann, R.; Wing, J. Automated generation and analysis of attack graphs. In *Proceedings 2002 IEEE Symposium on Security and Privacy*; IEEE: New York, NY, USA, 2002; pp. 273–284.
47. Ou, X.; Govindavajhala, S.; Appel, A.W. MulVAL: A logic-based network security analyzer. In *Proceedings of the 14th Conference on USENIX Security Symposium*; USENIX Association: Berkeley, CA, USA, 2005; p. 8.
48. Ou, X.; Boyer, W.F.; McQueen, M.A. A scalable approach to attack graph generation. In *CCS '06: Proceedings of the 13th ACM Conference on Computer and Communications Security*; Association for Computing Machinery (ACM): New York, NY, USA, 2006; pp. 336–345.
49. Hutchins, E.M.; Cloppert, M.J.; Amin, R.M. Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains. In *Leading Issues in Information Warfare and Security Research*; Ryan, J., Ed.; Academic Publishing International: Reading, UK, 2011; pp. 80–106.
50. Saint-Hilaire, K.A.; Neal, C.; Cuppens, F.; Cuppens-Boulahia, N.; Hadji, M. Matching Knowledge Graphs for Cybersecurity Countermeasures Selection. In *Proceedings of the 6th International Conference Science of Cyber Security (SciSec), Copenhagen, Denmark, 14–16 August 2024*; Springer Nature: Singapore, 2024
51. Saint-Hilaire, K.A.; Neal, C.; Cuppens, F.; Boulahia-Cuppens, N.; Bassi, F.; Hadji, M. A real-time automated attack-defense graph generation approach. *J. Inf. Secur. Appl.* **2025**, *94*, 104266. [CrossRef]
52. Merah, Y.; Kenaza, T. Ontology-Based Cyber Risk Monitoring Using Cyber Threat Intelligence. In *ARES '21: Proceedings of the 16th International Conference on Availability, Reliability and Security*; Association for Computing Machinery (ACM): New York, NY, USA, 2021.
53. AI-COMPLIANCE-AUDIT SCFPCI. 2026. Available online: https://github.com/AI-COMPLIANCE-AUDIT/SCF-PCI (accessed on 8 September 2026).
54. National Institute of Standards and Technology. *Open Security Controls Assessment Language (OSCAL)*; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2021.
55. ComplianceAsCode PCI DSS v4 OSCAL Catalog. 2026. Available online: https://github.com/ComplianceAsCode/oscal-content/blob/main/catalogs/pcidss_4/catalog.json (accessed on 8 September 2026).
56. Kent, K.; Chevalier, S.; Grance, T.; Dang, H. *Guide to Integrating Forensic Techniques into Incident Response*; NIST Special Publication 800-86; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2006.
57. Loumachi, F.Y.; Ghanem, M.C.; Ferrag, M.A. Advancing Cyber Incident Timeline Analysis Through Retrieval-Augmented Generation and Large Language Models. *Computers* **2025**, *14*, 67. [CrossRef]
58. Abdulaziz, M.; Koller, L. Formal Semantics and Formally Verified Validation for Temporal Planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*; AAAI Press: Palo Alto, CA, USA, 2022; Volume 36, pp. 9662–9670.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.