

Identifying Policy-Engine Semantics from 1 Distinguishing Probes: Sound Feasibility and Graded Verification for Outcome-Level Configuration Changes

Blessing Honmane

honmanehumbleoriola@gmail.com

Safelinx Technology <https://orcid.org/0009-0009-4362-321X>

Research Article

Keywords: policy engines, firewall configuration, active identification, assurance, verification, intent-based management

Posted Date: October 28th, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-11524043/v1>

License:  This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: The authors declare no competing interests.

Identifying Policy-Engine Semantics from Distinguishing Probes: Sound Feasibility and Graded Verification for Outcome-Level Configuration Changes

Blessing Honmane Director, Safelinx Technology
Lagos, Nigeria

Email: blessing.honmane@safelinx.ng

ORCID: <https://orcid.org/0009-0009-4362-321X>

Abstract: An agent asked to realize an outcome, such as permitting a set of flows, on an unfamiliar ordered-rule policy engine must know how the engine resolves overlapping rules. We ask how far sandbox probes can establish that when a finite class of candidate semantics is declared in advance. Probes identify the semantics exactly when they distinguish every pair of candidates, an instance of known oracle-guided identification. Our main finding is that identification does not show that the engine belongs to the class: a suite that leaves any modeled probe unqueried is blind to some out-of-class engine, and random probing needs $\Theta(1/\varepsilon)$ probes, up to a $\ln k$ factor, to detect an engine that is ε -far. On real Linux netfilter, Linux routing, nginx and OpenBSD pf, three or four probes identify in-class engines exactly, yet the same suite accepted a wrong hypothesis on all six out-of-class engines we built and on 123 more from a seeded generator. Structured and random suites detect different deviations, and no suite found the engines that deviate only at four rules. A plan can be checked with one probe per cell instead of one per flow: on a space of 65,536 flows this took 3 to 39 probes instead of 131,072, with the same verdicts. A sandbox that reuses connection-tracking state answers from history, which defeats identification and plan checks alike, and a control run catches it. AWS and Azure are simulated, not measured.

Index Terms: policy engines, firewall configuration, active identification, assurance, verification, intent-based management

I. INTRODUCTION

Outcome-level management interfaces let a client say what it wants, for example “permit these flows”, and leave the realization to an agent [1]. For ordered-rule policy engines the realization step is not mechanical. Whether a new permit rule takes effect depends on how the engine resolves overlapping rules, and engines differ: some honor the first matching rule, some the last, some an explicit priority, some the most specific match. An agent that assumes the wrong convention produces a configuration that looks correct and behaves incorrectly.

Example 1. Flows are the integers $\{0, 1, 2, 3\}$. The policy has one rule, $r_1 = ([1, 1], \text{deny})$, and the default is deny. The request is to permit $M = [0, 2]$. Under first-match semantics, inserting $([0, 2], \text{permit})$ at the top works. Under best-match semantics (the most specific matching rule wins), r_1 is strictly more specific than the new rule on flow 1, so no insertion

position makes flow 1 permitted. A planner that assumes first-match returns a plan that is wrong on flow 1.

The agent therefore needs evidence about the engine’s semantics and a clear statement of how far that evidence reaches. We formalize both and prove what can and cannot be established.

Contributions:

- 1) The price of class membership. A suite that leaves any modeled probe unqueried cannot certify that a target belongs to the declared class (Theorem 3), and random probing detects ε -far deviations with $\Theta(1/\varepsilon)$ probes, up to a $\ln k$ factor (Theorem 4, Corollary 1).
- 2) Measurements on four real engine families (Sections VII-D, VII-E, VII-F and VII-H): Linux netfilter, Linux routing, nginx and OpenBSD pf, driven with real packets or requests. In-class engines are identified exactly by suites of three or four probes. On six engines we built outside the class, the identification suite accepts a wrong hypothesis and only exhaustive or structured probing refutes it. A seeded generator, fixed before the run, then produced 123 further deviating engines, which test the structured suite on engines it was not designed around.
- 3) Oracle fidelity can fail silently. On Linux netfilter with the common established-connections accept, a sandbox that reuses source ports answers from earlier probes, which defeats both the model-based check and an apply-and-diff check. A control run catches it, and fresh source ports remove it (Section VII-G). A real-engine run on a space of 65,536 flows compares the cell check with enumeration (Section VII-A).
- 4) A hypothesis class drawn from vendor documentation, with simulations of iptables, pf, AWS network ACLs and Azure security groups, including a documented feature (pf’s quick) whose omission from the class yields a wrong plan, in simulation and on real OpenBSD pf (Sections VII and VII-F).
- 5) An assurance calculus with a soundness theorem and a fault-attribution corollary, which carries the identified semantics and its stated assumptions into a feasibility

verdict and graded verification (Sections V and VI).

Scope and novelty: A targeted literature check (Section II) found precedent for each ingredient. Theorems 1 and 2 are an instance of oracle-guided identification with distinguishing inputs [2] and of classical identification experiments [3]. Assumption 2 plays the role of the fault-domain assumption of conformance testing [4]. Elementary cells and exact differencing of rule lists are standard in firewall analysis [5], [6]. Choosing probes that separate every pair of candidates is a set-cover problem, as in work on identifying a language model behind a black-box interface [7]. The contribution is narrower: applying identification to the *conflict-resolution semantics* of a policy engine with configurations the agent installs in a sandbox; the explicit price of class membership (Theorems 3 and 4); and the way the identified semantics and its stated assumptions flow into a feasibility verdict and graded verification (Sections V and VI). The measurements cover four real engine families (Sections VII-D, VII-E, VII-F and VII-H). No commercial or cloud device is measured.

II. RELATED WORK

This survey is targeted, not exhaustive.

Outcome-level management and plans: Intent-based networking describes operators stating desired outcomes [1]. NETCONF and YANG give machine-readable configuration interfaces [8], [9]. Applying a saved Terraform plan executes only the operations recorded in the plan [10]; such tools rely on per-vendor provider code.

Verification under a fixed semantics model: Header-space analysis [11] and Batfish [12] derive the behavior that would result from a configuration; Zerkova encodes the semantics of the AWS policy language into SMT [13]. Firewall decision diagrams convert an ordered rule sequence into an equivalent conflict-free diagram [5], and diverse firewall design computes functional discrepancies between designs with such diagrams [6]. Policy-anomaly classification such as shadowing is treated in [14]. These works fix the semantics model in advance. Our Lemma 4 and Theorems 5 and 6 are standard in spirit for first-match lists; the point here is to attach them to an identified, assumption-labeled semantics.

Testing models and implementations by installed configurations: Metha generates configurations and compares the output of network analysis tools with real router software to find discrepancies in the tools’ models [15]. El-Atawy et al. test firewall enforcement with randomly generated policies and traffic [16]. These are closest to our probe model, in which the prober chooses the installed configuration. They check a given model for discrepancies; we select among candidate models and attach an assurance level to what follows.

Inferring policies by probing: Samak et al. reconstruct a fixed unknown firewall policy by active probing [17], [18]. Alhamwi et al. discover a firewall’s accept rules with a delay-based probing technique [19], and Salah et al. probe for the last-matching rules of a firewall [20]. Argyros et al. apply automata learning to black-box filters [21] and use black-box differential automata learning for evasion attacks and fingerprinting [22]. In these works the policy is fixed and the unknown is the policy

itself; here the prober installs policies and the unknown is the engine’s conflict-resolution semantics, drawn from a declared class. Iyer and Masoumzadeh learn the policy enforced by a black-box access-control system from its responses [23]; there too the unknown is the policy.

Authorization languages and combining algorithms:

Authorization languages usually fix their conflict resolution by design. Cedar lets forbid policies override permit policies and denies by default, and its semantics is formalized and tested against a reference model [24], [25]. XACML defines combining algorithms such as deny-overrides and first-applicable, with a formal semantics [26] and conditions under which pairs of algorithms differ [27]; Margrave computes semantic differences between XACML policies [28]. These works take the resolution rule as given. The pairwise difference conditions are the kind of fact a distinguishing probe relies on, and the combining algorithms are natural members of a hypothesis class.

Validating models against real devices: CrystalNet runs production switch software in an emulated network instead of modeling it [29]. Krentsel et al. argue that hand-written models of router control planes cannot stay faithful, and report a configuration-ordering assumption in Batfish’s model that disagreed with emulated routers [30]. This is evidence that ordering semantics is a real unknown. These systems replace the model by the device; they do not infer, select or certify a semantics.

Model inference for network functions: Alembic [31] infers behavioral models of black-box firewalls, NATs and load balancers by installing configurations and learning, for each rule type, a symbolic finite-state machine with an extension of Angluin’s algorithm [32]. It takes the rule-processing semantics (first-match in its implementation) from vendor documentation as an input, and its evaluation concerns stateful TCP behavior. We study the complementary stateless question: the conflict-resolution semantics is the unknown, drawn from a declared class, and we characterize what a probe suite can and cannot establish about it. An identified semantics could serve as Alembic’s processing-semantics input.

Identification theory: Identifying a member of a finite class from query answers is classical [3], [32]. In conformance testing, finite test suites are justified only relative to a fault-domain assumption [4], which corresponds to our Assumption 2; Theorem 3 is a formal statement of why it cannot be dropped. Oracle-guided synthesis keeps a set of candidates consistent with oracle answers and asks for a distinguishing input between non-equivalent candidates until one remains [2]; Theorems 1 and 2 and Algorithm 1 are an instance of that scheme for a finite class. Das and Gupta formalize identification of one of M candidate language models behind a black-box interface and show that choosing probes so that every pair is separated is exactly weighted set cover [7]; their responses are probabilistic where ours are deterministic, and their candidates are models rather than conflict-resolution rules, but the pairwise-separation structure is the same.

Untrusted producer, trusted checker: Proof-carrying code [33] and translation validation [34] separate an untrusted producer from a small trusted checker; Section VI applies that separation to proposals of semantics and plans.

Gap: Probing a network function with installed configurations to infer a model is established [31]. To our knowledge, no work treats the conflict-resolution semantics of a policy engine as the unknown drawn from a declared class and combines (i) its exact identification conditions, (ii) a quantified account of what class membership costs, and (iii) the propagation of that conditional into feasibility and verification claims. The nearest works are black-box policy learning [23], which infers the policy and not the semantics, and model-free verification [30], which motivates the problem but does not select among semantics.

III. MODEL

A. Flows, rules, policies

Let $X_j = \{0, \dots, N_j - 1\}$ for $j = 1, \dots, d$ and let the *flow space* be $F = X_1 \times \dots \times X_d$ (finite). A *box* is $B = \prod_{j=1}^d [a_j, b_j]$ with $0 \leq a_j \leq b_j < N_j$; write $|B|$ for its cardinality. Decisions are $\mathbb{D} = \{0, 1\}$, with $1 = \text{permit}$ and $0 = \text{deny}$.

A *rule* is $r = (B, a, \pi)$ with box B , action $a \in \mathbb{D}$ and priority $\pi \in \mathbb{N}$. A *policy* is $P = (r_1, \dots, r_n; \delta)$ with $n \geq 0$ rules and default $\delta \in \mathbb{D}$; write $r_i = (B_i, a_i, \pi_i)$ and $[n] = \{1, \dots, n\}$. Let $\mathcal{P}$ be the set of all policies. The *signature* of a flow is

$$\sigma_P(f) = \{i \in [n] : f \in B_i\}.$$

B. Rule-list semantics

A *semantics* h is given by a *winner map* $w_h(P, S) \in S$ for each policy P and nonempty $S \subseteq [n]$. It defines the decision function

$$\text{ev}_h(P, f) = \begin{cases} a_{w_h(P, \sigma_P(f))} & \text{if } \sigma_P(f) \neq \emptyset, \\ \delta & \text{otherwise.} \end{cases}$$

Four semantics are used throughout:

$$\begin{aligned} w_{\text{FM}}(P, S) &= \min S, & w_{\text{LM}}(P, S) &= \max S, \\ w_{\text{PR}}(P, S) &= \min\{i \in S : \pi_i = \pi_S\}, & \pi_S &= \max_{j \in S} \pi_j, \\ w_{\text{BM}}(P, S) &= \min\{i \in S : |B_i| = v_S\}, & v_S &= \min_{j \in S} |B_j|. \end{aligned}$$

These are first-match, last-match, highest-priority (ties to the earlier rule) and most-specific-match (smaller box wins, ties to the earlier rule). They are stylized; real engines have further features (zones, address translation, state, application identification). Assumption 4 below makes that gap explicit.

Lemma 1 (Signature determinacy). *For every semantics h as above, if $\sigma_P(f) = \sigma_{P'}(f')$ then $\text{ev}_h(P, f) = \text{ev}_h(P', f')$.*

Proof. Immediate from the definition: $\text{ev}_h(P, f)$ depends on f only through $\sigma_P(f)$. $\square$

C. Sandbox oracle and probes

Let $\mathcal{P}^* \subseteq \mathcal{P}$ be the class of policies the model can express, and $\mathcal{U}^* = \mathcal{P}^* \times F$ the *modeled domain*. The target is a function $T : \mathcal{U}^* \rightarrow \mathbb{D}$. A *probe* is a pair $u = (P, f)$; the sandbox lets the agent install P and query f . Let $\mathcal{U} \subseteq \mathcal{U}^*$ be the *probe universe* the sandbox can execute. A finite *hypothesis class* $\mathcal{H}$ with $|\mathcal{H}| = k$ is declared in advance. It is *irredundant on $\mathcal{U}$*

TABLE I
ASSUMPTIONS USED BY EACH RESULT. O: ORACLE FIDELITY, C: CLASS MEMBERSHIP, S: SIGNATURE DETERMINACY, M: MODELABILITY. THEOREM 3 AND THE SOUNDNESS THEOREM USE NONE; THEOREM 4 USES PROBES DRAWN INDEPENDENTLY OF THE TARGET.

Result	O	C	S	M
Thm. 1(a), identification	x	x		
Thm. 3, blind targets				
Thm. 4, detection	x			
Prop. 2, planning		x		x
Thm. 5, Q1 by cells	x		x	x
Thm. 6, non-interference		x		x
Thm. 7, soundness				

if for all $h \neq h'$ in $\mathcal{H}$ there is $u \in \mathcal{U}$ with $\text{ev}_h(u) \neq \text{ev}_{h'}(u)$ (observationally equal members are merged).

Assumption 1 (Oracle fidelity). For every $u \in \mathcal{U}$ the sandbox returns $T(u)$, deterministically.

Assumption 2 (Class membership). There exists $h^* \in \mathcal{H}$ with $T(u) = \text{ev}_{h^*}(u)$ for all $u \in \mathcal{U}^*$.

Assumption 3 (Signature determinacy). For every $P \in \mathcal{P}^*$, $T(P, f) = T(P, f')$ whenever $\sigma_P(f) = \sigma_P(f')$.

Assumption 4 (Modelability). The production policy, and every policy the agent proposes, lie in $\mathcal{P}^*$.

Assumption 2 implies Assumption 3 (by Lemma 1) but not conversely. We keep them apart because Section V shows some results need only the weaker one. Table I lists which result uses which assumption.

IV. IDENTIFICATION BY DISTINGUISHING PROBES

Definition 1 (Suites, consistency). A finite set $S \subseteq \mathcal{U}$ is a *distinguishing suite* for $\mathcal{H}$ if for all $h \neq h'$ in $\mathcal{H}$ there is $u \in S$ with $\text{ev}_h(u) \neq \text{ev}_{h'}(u)$. The *consistent set* of S for target T is

$$\text{Cons}(S, T) = \{h \in \mathcal{H} : \forall u \in S, \text{ev}_h(u) = T(u)\}.$$

Theorem 1 (Identification). *Let $\mathcal{H}$ be irredundant on $\mathcal{U}$ and $S \subseteq \mathcal{U}$ finite.*

- (a) *If S is distinguishing and T satisfies Assumption 2 with witness h^* , then $\text{Cons}(S, T) = \{h^*\}$.*
- (b) *If S is not distinguishing, there is a target satisfying Assumption 2 with $|\text{Cons}(S, T)| \geq 2$.*

Hence S identifies every in-class target if and only if S is distinguishing.

Proof. (a) $h^* \in \text{Cons}(S, T)$ because $T = \text{ev}_{h^*}$ on $\mathcal{U}^* \supseteq S$. Suppose $h \neq h^*$ lay in $\text{Cons}(S, T)$. Then $\text{ev}_h(u) = T(u) = \text{ev}_{h^*}(u)$ for all $u \in S$, contradicting that S separates h and h^* . (b) Choose $h \neq h'$ not separated by S and let $T = \text{ev}_h$. Then T is in-class, and $\text{ev}_{h'}(u) = \text{ev}_h(u) = T(u)$ for all $u \in S$, so $\{h, h'\} \subseteq \text{Cons}(S, T)$. $\square$

Proposition 1 (Agreement). *Assume Assumptions 1 and 2, and let $C = \text{Cons}(S, T)$ for any finite $S \subseteq \mathcal{U}$ (distinguishing or not). If $\text{ev}_h(P', f) = \text{ev}_{h'}(P', f)$ for all $h, h' \in C$, then $T(P', f) = \text{ev}_h(P', f)$ for every $h \in C$.*

Algorithm 1 Deterministic greedy distinguishing suite

Require: $\mathcal{H}$; ordered probe pool $(u_1, \dots, u_N) \subseteq \mathcal{U}$

- 1: $S \leftarrow ()$; $\Pi \leftarrow \{\mathcal{H}\}$
- 2: **while** some class $K \in \Pi$ has $|K| > 1$ **do**
- 3: $t \leftarrow$ least index such that u_t splits some class of Π
- 4: **if** no such t exists **then**
- 5: **return** FAIL (two members are observationally equal on the pool)
- 6: **end if**
- 7: append u_t to S ; refine Π by the responses at u_t
- 8: **end while**
- 9: **return** S

Proof. By Assumption 2 and $T = \text{ev}_{h^*}$ on S , $h^* \in C$, and $T(P', f) = \text{ev}_{h^*}(P', f) = \text{ev}_h(P', f)$ for every $h \in C$ by agreement. $\square$

Proposition 1 lets a conclusion about a specific policy stand even when the suite leaves several hypotheses consistent, provided they agree on that policy.

Theorem 2 (Size of distinguishing suites). *Let $|\mathcal{H}| = k \geq 1$ with $\mathcal{H}$ irredundant on $\mathcal{U}$.*

- (a) *Every distinguishing suite has $|S| \geq \lceil \log_2 k \rceil$.*
- (b) *A distinguishing suite with $|S| \leq k - 1$ exists, and Algorithm 1 constructs one deterministically whenever its probe pool separates every pair of hypotheses.*
- (c) *For every $k \geq 2$ there exist classes attaining the lower bound and classes attaining the upper bound.*

Proof. (a) For a suite S define $\varphi_S : \mathcal{H} \rightarrow \mathbb{D}^S$ by $\varphi_S(h) = (\text{ev}_h(u))_{u \in S}$. S is distinguishing iff φ_S is injective. Therefore $k \leq 2^{|S|}$.

(b) Algorithm 1 maintains the partition Π of $\mathcal{H}$ into classes with equal response vectors on the current suite. Each iteration adds a probe that splits at least one class, so $|\Pi|$ strictly increases; it starts at 1 and cannot exceed k , so there are at most $k - 1$ iterations. If some class has two members, irredundance gives a probe in $\mathcal{U}$ separating them, so a splitting probe exists and the algorithm does not fail. At termination every class is a singleton, so φ_S is injective.

(c) *Lower bound attained.* Let $m = \lceil \log_2 k \rceil$, $\mathcal{U} = \{u_1, \dots, u_m\}$, and let $h_0, \dots, h_{k-1}$ be defined by $\text{ev}_{h_i}(u_j) =$ the j -th bit of the binary expansion of i . The k vectors are distinct, so $\mathcal{H}$ is irredundant and $S = \mathcal{U}$ is distinguishing with $|S| = m$. *Upper bound attained.* Let $\mathcal{U} = \{u_1, \dots, u_{k-1}\}$, $\text{ev}_{h_0} \equiv 0$ and $\text{ev}_{h_i}(u_j) = [i = j]$ for $i \geq 1$. Then h_0 and h_i differ only at u_i , so every distinguishing suite contains every u_i , and the minimum size is $k - 1$. $\square$

The output depends only on the ordered pool and $\mathcal{H}$, as a deterministic checker requires. It is not minimum in general: a minimum suite is a minimum set cover over pairs of hypotheses.

A. What class membership costs

Theorem 3 (Blind targets). *Let S be a distinguishing suite for $\mathcal{H}$ and $u \in \mathcal{U}^* \setminus S$. For $h \in \mathcal{H}$ define $T^{h,u}(x) = \text{ev}_h(x)$ for $x \neq u$ and $T^{h,u}(u) = 1 - \text{ev}_h(u)$. Then*

- (i) $\text{Cons}(S, T^{h,u}) = \{h\}$, and
- (ii) *there is no $h' \in \mathcal{H}$ with $\text{ev}_{h'} = T^{h,u}$ on $\mathcal{U}^*$.*

Proof. (i) $T^{h,u} = \text{ev}_h$ on S since $u \notin S$, so $h \in \text{Cons}$. Any $h' \neq h$ in Cons would agree with ev_h on S , contradicting that S is distinguishing. (ii) Suppose $\text{ev}_{h'} = T^{h,u}$ on $\mathcal{U}^*$. Since $u \notin S$, $\text{ev}_{h'} = \text{ev}_h$ on S , so $h' = h$ because S is distinguishing. But then $\text{ev}_h(u) = T^{h,u}(u) = 1 - \text{ev}_h(u)$, a contradiction. $\square$

Theorem 3 says that every distinguishing suite that omits some modeled probe identifies some out-of-class target as a member of $\mathcal{H}$, and that the identified member is wrong at u . The probes therefore cannot discharge Assumption 2, and any conclusion that depends on it must carry it explicitly. The theorem concerns suites that leave some modeled probe unqueried. If the modeled domain is finite and small enough to query in full, as in the exhaustive sweeps of Section VII, the whole domain is a suite and does certify membership for that domain; with policies of unbounded length no finite suite does. Detecting deviations is a separate, statistical question.

Theorem 4 (Detection by random probing). *Let μ be a probability distribution on a finite pool $\mathcal{U}_0 \subseteq \mathcal{U}^*$ and $T : \mathcal{U}_0 \rightarrow \mathbb{D}$ any target. For $h \in \mathcal{H}$ let $d_h = \mu\{u : T(u) \neq \text{ev}_h(u)\}$ and $\varepsilon = \min_{h \in \mathcal{H}} d_h$. Let $S = (u_1, \dots, u_n)$ be drawn i.i.d. from μ , independently of T . Then*

$$(1 - \varepsilon)^n \leq \Pr[\text{Cons}(S, T) \neq \emptyset] \leq \sum_{h \in \mathcal{H}} (1 - d_h)^n \leq k e^{-\varepsilon n}.$$

Proof. $h \in \text{Cons}(S, T)$ iff every u_i avoids $\{u : T(u) \neq \text{ev}_h(u)\}$, which has probability $(1 - d_h)^n$. The union bound gives the middle inequality. For the lower bound take h_0 with $d_{h_0} = \varepsilon$: $\Pr[h_0 \in \text{Cons}] = (1 - \varepsilon)^n$. Finally $(1 - d_h)^n \leq (1 - \varepsilon)^n \leq e^{-\varepsilon n}$. $\square$

Say that T is ε -far from $\mathcal{H}$ when $\min_h d_h \geq \varepsilon$ and that deviation is *detected* when $\text{Cons}(S, T) = \emptyset$.

Corollary 1 (Sample complexity). *For $0 < \varepsilon < 1$ and $0 < \delta < 1$: (i) $n \geq \varepsilon^{-1} \ln(k/\delta)$ suffices for detection with probability at least $1 - \delta$ against every ε -far T ; (ii) if some T has $\min_h d_h = \varepsilon$, then detection with probability at least $1 - \delta$ requires $n \geq (1 - \varepsilon) \varepsilon^{-1} \ln(1/\delta)$.*

Proof. (i) From Theorem 4, $\Pr[\text{miss}] \leq k e^{-\varepsilon n} \leq \delta$. (ii) A miss has probability at least $(1 - \varepsilon)^n$, so we need $(1 - \varepsilon)^n \leq \delta$, i.e. $n \geq \ln(1/\delta) / \ln \frac{1}{1 - \varepsilon}$, and $\ln \frac{1}{1 - \varepsilon} \leq \frac{\varepsilon}{1 - \varepsilon}$. $\square$

Remark 1 (Determinism and adversaries). Theorem 4 treats T as fixed independently of the draw. A deterministic suite generated from a fixed public seed carries this guarantee only against targets whose deviations are independent of the seed. Against a target that adapts to the suite, Theorem 3 applies: no deterministic suite that omits a modeled probe detects all deviations. An adversary who knows the suite, such as a hook rule installed to answer the known probes correctly, can therefore evade it. The bounds of Theorem 4 hold against such an adversary only if the probes are drawn from a source the adversary cannot see.

Remark 2 (Identification and detection are different costs). Identification of an in-class target needs at most $k - 1$ probes,

a quantity independent of the target and of $\mathcal{U}$ (Theorem 2). Detecting an ε -far target by random probing needs on the order of $1/\varepsilon$ probes. Structured deviations may be found with fewer probes by targeted tests; that is a design question for the probe pool, not something these bounds exclude.

V. FROM IDENTIFICATION TO VERIFICATION

Throughout this section T is the target, P' is the policy after a proposed change, and M is the requested match set, a box. The two postconditions are: (Q1) $T(P', f) = \text{permit}$ for all $f \in M$; (Q2) $T(P', f) = T(P, f)$ for all $f \notin M$, where P is the policy before the change.

A. The verdict depends on the semantics

Let $P' = (r_1, \dots, r_p, r, r_{p+1}, \dots, r_n; \delta)$ denote P with a rule r inserted at position $p \in \{0, \dots, n\}$; the new rule has index $p+1$. Let $\iota_p : [n] \rightarrow [n+1]$ map $i \mapsto i$ for $i \leq p$ and $i \mapsto i+1$ otherwise. A semantics h is *insertion-stable* if $w_h(P', \iota_p(S)) = \iota_p(w_h(P, S))$ for all nonempty $S \subseteq [n]$, all rules r and all positions p .

Lemma 2. FM, LM, PR, BM are insertion-stable.

Proof. Rule i of P is rule $\iota_p(i)$ of P' with the same priority and box, and ι_p is strictly increasing, hence preserves the index order used in all four winner maps and in their tie-breaking. $\square$

Lemma 3 (Locality). *If h is insertion-stable and P' is P with $r = (B, a, \pi)$ inserted, then $\text{ev}_h(P', f) = \text{ev}_h(P, f)$ for every $f \notin B$.*

Proof. For $f \notin B$ the new rule does not match, so $\sigma_{P'}(f) = \iota_p(\sigma_P(f))$. If this set is empty both decisions are δ . Otherwise insertion stability gives $w_h(P', \sigma_{P'}(f)) = \iota_p(w_h(P, \sigma_P(f)))$, which is the same rule. $\square$

Proposition 2 (Planning and semantic dependence). *Let M be a box and $r = (M, \text{permit}, \pi)$.*

- (a) *Under FM, inserting r at position 0 satisfies Q1; under LM, inserting at position n does; under PR, inserting at any position with $\pi > \max_i \pi_i$ does.*
- (b) *In each case Q2 holds as well.*
- (c) *Under BM there exist P and M for which no insertion of a rule with box M , at any position and priority, satisfies Q1.*

Proof. (a) For $f \in M$ the new rule is in the signature. Under FM at position 0 it has the least index; under LM at position n the greatest; under PR it has strictly the greatest priority. In each case it wins, and its action is permit. (b) Lemma 3 with $B = M$. (c) Example 1: $r_1 = ([1, 1], \text{deny})$ has $|B_1| = 1 < 3 = |M|$, so r_1 beats r at flow 1 for every position. Priorities are ignored by BM. $\square$

Proposition 2 shows the consequence of misidentification. In Example 1 the plan “insert at position 0” is correct under FM and false under BM. A checker that reports a feasible plan on the basis of an unidentified semantics can therefore report a false verdict, which is why identification evidence must precede the feasibility verdict.

B. Verifying universal postconditions

Lemma 4 (Elementary cells). *Let $\mathcal{B}$ be a set of m boxes in F . There is a partition of F into at most $(2m+1)^d$ boxes (cells) such that every $B \in \mathcal{B}$ either contains a cell or is disjoint from it. In particular, σ_P is constant on each cell when the rules of P have boxes in $\mathcal{B}$.*

Proof. For each coordinate j let $K_j = \{0\} \cup \{a_j(B) : B \in \mathcal{B}\} \cup \{b_j(B) + 1 : B \in \mathcal{B}, b_j(B) + 1 < N_j\}$; then $|K_j| \leq 2m+1$. The points of K_j cut X_j into $|K_j|$ maximal segments $[s, t]$ with no cut point in $(s, t]$. For $x, x+1$ in one segment, $x+1 \notin K_j$, so $x+1 \neq a_j(B)$ and $x+1 \neq b_j(B)+1$, hence $x+1 \in [a_j(B), b_j(B)]$ iff $x \in [a_j(B), b_j(B)]$: membership is constant on a segment. A flow lies in B iff each coordinate lies in the corresponding interval, so membership of B is constant on every product of segments. There are at most $\prod_j |K_j| \leq (2m+1)^d$ such products. $\square$

Theorem 5 (Partition verification of Q1). *Assume Assumption 3 and $P' \in \mathcal{P}^*$ with n' rules. Let $\mathcal{C}$ be the cells of Lemma 4 for the boxes of P' together with M , and let R contain one flow from each cell contained in M . Then $|R| \leq (2(n'+1)+1)^d$, and if $T(P', q) = \text{permit}$ for all $q \in R$ then $T(P', f) = \text{permit}$ for all $f \in M$.*

Proof. Since M is among the boxes, every cell is contained in M or disjoint from it, and the cells contained in M cover M . By Lemma 4, $\sigma_{P'}$ is constant on each cell, so by Assumption 3 $T(P', \cdot)$ is constant on each cell. The count follows from Lemma 4 with $m = n' + 1$. $\square$

Remark 3. Theorem 5 uses only signature determinacy, not the identification of h^* . Identification is needed for what comes before and after: choosing where to insert (Proposition 2) and predicting the effect on flows outside M without querying the target (Theorem 6). If Assumption 3 is not available, probing a cell representative establishes only that representative, and the postcondition is verified partially.

Theorem 6 (Non-interference). *Assume Assumptions 2 and 4, with witness h^* insertion-stable. Let P' be P with $r = (B, \text{permit}, \pi)$ inserted, $M \subseteq B$ a box, and $\Delta = \{f : \text{ev}_{h^*}(P, f) \neq \text{ev}_{h^*}(P', f)\}$. Then:*

- (i) $\Delta \subseteq B$; hence if $B = M$, Q2 holds.
- (ii) *In general Q2 holds iff $\Delta \subseteq M$, and this is decidable in time $O(n(2n+5)^d)$, where $n = |P|$.*

Proof. Under Assumption 2, $T = \text{ev}_{h^*}$ on $\mathcal{U}^*$, and $P, P' \in \mathcal{P}^*$ by Assumption 4. (i) is Lemma 3. (ii) Q2 states $\Delta \subseteq M$ by definition. Take the cells of Lemma 4 for the boxes of P' together with M ; since the boxes of P are among those of P' , both $\text{ev}_{h^*}(P, \cdot)$ and $\text{ev}_{h^*}(P', \cdot)$ are constant on each cell (Lemma 1). Hence Δ is a union of cells, and every cell is inside M or disjoint from it. Q2 holds iff every cell disjoint from M has equal decisions under P and P' . There are at most $(2(n+2)+1)^d = (2n+5)^d$ cells, and each evaluation costs $O(n)$. $\square$

Remark 4 (Exact fidelity). Theorem 6(i) turns non-interference into a binding obligation: the inserted rule’s box must equal the requested match. If the native object resolves to a strictly larger

box (a “broader” realization), Q2 can fail, and (ii) computes exactly where. The time bound is for direct enumeration of cells; decision diagrams avoid it.

VI. AN ASSURANCE CALCULUS

A. Levels, evidence, rules

Let $\mathbb{L} = \{U < A < S < E\}$ (unknown, asserted, supported, established), a chain with meet min. Let Ω be a set of worlds. Each *claim* c has a truth set $\llbracket c \rrbracket \subseteq \Omega$. A finite set $\mathcal{A}$ of named *assumptions* (for example Assumptions 1–4 and platform assumptions such as the correctness of native authorization) has truth sets $\llbracket \tau \rrbracket \subseteq \Omega$; for $A \subseteq \mathcal{A}$ let $\Omega_A = \bigcap_{\tau \in A} \llbracket \tau \rrbracket$.

An *evidence item* e has a claim $c(e)$, a level $\lambda(e) \in \mathbb{L}$ and an assumption set $\alpha(e) \subseteq \mathcal{A}$. It is *base-sound* if $\lambda(e) = E$ implies $\Omega_{\alpha(e)} \subseteq \llbracket c(e) \rrbracket$.

A *rule* is $\rho = (\Gamma_\rho, c_\rho, m_\rho, \alpha_\rho)$: premise claims Γ_ρ , conclusion c_ρ , cap $m_\rho \in \mathbb{L}$, assumptions $\alpha_\rho \subseteq \mathcal{A}$. It is *E-sound* if $m_\rho = E$ implies

$$\Omega_{\alpha_\rho} \cap \bigcap_{\gamma \in \Gamma_\rho} \llbracket \gamma \rrbracket \subseteq \llbracket c_\rho \rrbracket.$$

A *derivation* is a finite tree whose leaves are evidence items and whose internal nodes apply rules to their children’s conclusions. Node levels and assumption sets are defined by

$$\begin{aligned} \lambda(v) &= \min(m_\rho, \min_{v' \text{ child of } v} \lambda(v')), \\ \alpha(v) &= \alpha_\rho \cup \bigcup_{v' \text{ child of } v} \alpha(v'). \end{aligned}$$

Proposals from an untrusted source enter as leaves with level U and are raised only by rules that consume independent evidence.

Theorem 7 (Soundness of established claims). *If every evidence item is base-sound and every rule is E-sound, then for every derivation node v with $\lambda(v) = E$ and conclusion c_v ,*

$$\Omega_{\alpha(v)} \subseteq \llbracket c_v \rrbracket.$$

Proof. Induction on the tree. A leaf is base-sound. For an internal node v with rule ρ and $\lambda(v) = E$ we have $m_\rho = E$ and all children at E . By induction, $\Omega_{\alpha(v')} \subseteq \llbracket \gamma \rrbracket$ for each child v' with conclusion $\gamma \in \Gamma_\rho$. Since $\alpha(v) \supseteq \alpha_\rho \cup \bigcup \alpha(v')$, $\Omega_{\alpha(v)} \subseteq \Omega_{\alpha_\rho} \cap \bigcap_{\gamma \in \Gamma_\rho} \llbracket \gamma \rrbracket \subseteq \llbracket c_\rho \rrbracket$ by E-soundness. $\square$

Proposition 3 (No laundering). *For every node v , $\lambda(v) \leq \lambda(\ell)$ for every leaf ℓ below v , and $\lambda(v) \leq m_\rho$ for every rule ρ applied at or below v .*

Proof. Induction: $\lambda(v)$ is a minimum that includes m_ρ and the children’s levels. $\square$

Consequently a claim whose derivations all rest on a declaration capped at A cannot be established, however many rules are applied.

Corollary 2 (Fault attribution). *Suppose a derivation node v with $\lambda(v) = E$ and conclusion c_v , built from base-sound evidence and E-sound rules. If $\omega \in \Omega$ and $\omega \notin \llbracket c_v \rrbracket$, then $\omega \notin \llbracket \tau \rrbracket$ for some $\tau \in \alpha(v)$.*

Proof. Contrapositive of Theorem 7. $\square$

Corollary 2 is what makes a failure traceable: a wrongly established claim is attributed to a named assumption in $\alpha(v)$ (which the result discloses), or else to an unsound rule or base item, which is a defect of the specification rather than of the world.

B. Instantiating the calculus

Semantics identification: Let ρ_{sem} have conclusion “ $T = \text{ev}_h$ on $\mathcal{U}^{**}$ ”, premises the E-level observations $T(u)$ for $u \in S$ (under Assumption 1), the decidable fact that S is distinguishing for $\mathcal{H}$, and h the unique element of $\text{Cons}(S, T)$. Its assumption set is $\{\text{Assumption 1, Assumption 2}\}$. By Theorem 1(a), ρ_{sem} is E-sound. Theorem 3 shows the assumption on class membership cannot be dropped.

Universal verification: The partition rule has conclusion Q1, premises the probe responses on R and the cell construction, and assumptions $\{\text{Assumption 1, Assumption 3}\}$. It is E-sound by Theorem 5. The differencing rule for Q2 additionally needs Assumptions 2 and 4 and is E-sound by Theorem 6. Without them the results are capped below E and the reported verification status is partial.

Feasibility: A request is *feasible* at the stated model version iff every precondition claim of the requested profile has a derivation at E , the plan’s dependencies satisfy their guards, and no unresolved conflict exists. Let A^* be the union of the derivations’ assumption sets. Under Corollary 2, every precondition holds at assessment time in every world satisfying A^* . Feasible does not mean the change succeeds in the live system: execution can fail, concurrent changes can intervene, and verification can be partial.

A worked request: Take Example 1 with $\mathcal{H} = \{\text{FM}, \text{BM}\}$. The suite is one probe $u_1 = (P_0, 2)$ with $P_0 = ([0, 3], \text{permit}), ([2, 2], \text{deny})$, which is distinguishing: FM permits flow 2 and BM denies it. The plan inserts $r = ([0, 2], \text{permit})$ at position 0, giving P' . By Theorem 5 the cells inside $M = [0, 2]$ are the flows 0, 1, 2, so Q1 is checked by three probes of $(P', \cdot)$. Since the box of r equals M , Theorem 6(i) gives Q2 without further probes.

- *In-class target, first-match.* $T(u_1) = \text{permit}$, so $\text{Cons} = \{\text{FM}\}$. The three probes return permit. Q1 is established at E under Assumptions 1 and 3, and Q2 at E under Assumptions 2 and 4 as well.
- *Out-of-class target that is first-match except that $(P', 1)$ is denied* (a hidden hook; this is $T^{\text{FM}, u}$ of Theorem 3 with $u = (P', 1) \notin S$). The identification suite still returns $\{\text{FM}\}$ and the plan is still chosen. The Q1 probe at flow 1 returns deny, so Q1 is not established and the change is rejected. The calculus did not detect the deviation by identification; the probe in the Q1 check did, because Theorem 5 does not rely on class membership.
- *Out-of-class target that is first-match except that $(P', 3)$ is permitted.* Flow 3 is outside M . Q1 passes, and Q2 is reported at E because the model predicts no change at flow 3. Q2 is false. Corollary 2 attributes the error to Assumption 2, which is in the assumption set of the Q2 claim and was disclosed with it. Nothing in the procedure detects it unless flow 3 is probed.

TABLE II

PROBES NEEDED TO CHECK A PLAN, $N = 256$, $M = [64, 191]^d$, ONE FIXED RANDOM EXAMPLE PER ROW. “ENUMERATE” QUERIES EVERY FLOW, “CELLS” ONE FLOW PER ELEMENTARY CELL OF P' AND M , “Q1 CELLS” ONLY THE CELLS INSIDE M .

d	n	Enumerate	Cells	Q1 cells
1	5	256	12	3
1	10	256	23	11
2	5	65,536	169	18
2	10	65,536	441	80
3	5	16,777,216	1,728	18
3	10	16,777,216	9,200	352

VII. COMPUTATIONAL CHECKS AND MEASUREMENTS

This section compares the cell check with enumeration, simulates documented vendor behavior, and then measures real engines in a sandbox. Computational checks of the theorems on small instances are in the appendix.

A. A baseline: apply and diff

The obvious alternative to identification is to install the planned policy in the sandbox and query every flow, before and after. This needs no hypothesis class and no class assumption, and it checks Q1 and Q2 on the real engine for that plan. Where F can be enumerated it is the stronger test, and it would have caught both out-of-class cases of the worked request in Section VI. The cell check has an advantage only in cost. Table II counts probes (script `baseline_cost.py`) for $N = 256$ values per dimension, a request box $M = [64, 191]^d$ and n random rules. Enumeration needs N^d probes for Q1 and Q2, while probing one flow per cell needs a number that depends on n and d but not on N . Probing every cell of P' and of P checks Q1 and Q2 with Assumption 3 only, and it is the comparison to make when F is too large to enumerate. Using the identified semantics to predict Q2 instead (Theorem 6) leaves only the Q1 probes, at the price of Assumption 2. The measurements here count probes only and do not time anything.

We also ran both checks on real nftables with a flow space of 65,536 addresses (script `real_engine_experiment8_scale.py`, kernel 6.18.44). Flow f is the destination address $10.98.(f \gg 8).(f \& 255)$, one socket receives every datagram, and a plan inserts a permit rule for a random box M of 8,000 to 30,000 addresses at position 0 of a random policy of 5 to 40 rules. For each of eight plans we installed the old and the new policy and queried all 65,536 flows under each (131,072 probes, about 11 s per plan), then probed one flow per cell inside M (3 to 39 probes). The verdicts for Q1 and Q2 agreed on all eight plans, and the engine equalled first-match on every flow. The cell check took a fraction of a second, but that time is dominated by a fixed settling wait and the rule installation, so we report probe counts rather than a speed-up. The cell check relies on signature determinacy: a second table that drops one address inside M , not at a cell boundary, passed the cell check with one probe and failed enumeration. The cell check is therefore exact for what it covers and silent about a deviation inside a cell.

The gap grows with d because the cell count is bounded by $(2n+5)^d$ regardless of N . For one dimension and a few hundred flows enumeration is simpler and safer, and we would use it. The results of this paper therefore matter most when the flow space is large, and when the agent needs the identified semantics anyway to choose where to insert the rule (Proposition 2).

B. A hypothesis class from vendor documentation

Table III records, for four engines, the documented conflict-resolution behavior and the member of the class that models it. We take the class $\mathcal{H}_8 = \{\text{FM}, \text{LM}, \text{PR}^\uparrow, \text{PR}^\downarrow, \text{BM}, \text{PO}, \text{DO}, \text{LMQ}\}$. Here $\text{PR}^\uparrow$ and $\text{PR}^\downarrow$ are priority with the higher or the lower value winning (the direction is not uniform across engines), PO and DO are permit-overrides and deny-overrides, and LMQ is last-match in which a rule marked *quick* ends evaluation. BM, PO and DO are included to test discrimination; no vendor claim is made for them in this paper. Rules in $\mathcal{H}_8$ carry a quick flag $q_i \in \{0, 1\}$ that the other semantics ignore. With priorities and tie-breaking as in Section III, the winner maps are $w_{\text{PR}^\uparrow} = w_{\text{PR}}$, $w_{\text{PR}^\downarrow}(P, S) = \min\{i \in S : \pi_i = \min_{j \in S} \pi_j\}$, $w_{\text{PO}}(P, S) = \min\{i \in S : a_i = 1\}$ if that set is nonempty and $\min S$ otherwise, w_{DO} likewise with $a_i = 0$, and $w_{\text{LMQ}}(P, S) = \min\{i \in S : q_i = 1\}$ if that set is nonempty and $\max S$ otherwise.

Lemma 5. *Every member of $\mathcal{H}_8$ is insertion-stable.*

Proof. Each winner map picks an index from S by comparing attributes of the rules (action, priority, quick flag, box size) and breaking ties by the order of the indices, through min or max. The map ι_p is strictly increasing, and rule $\iota_p(i)$ of P' has the same attributes as rule i of P . The comparisons and the tie-breaks therefore select $\iota_p(w_h(P, S))$. The new rule has index $p + 1 \notin \iota_p(S)$ and does not enter. $\square$

Lemma 5 lets Theorem 6 and Proposition 2(b) apply to a target in any member of $\mathcal{H}_8$. We also checked it by brute force (script `check_insertion_stability.py`) over all policies of one or two rules, all inserted rules, positions and index sets, which is 1.2×10^8 cases with no violation.

C. Identification on simulated engines

The script `engine_experiments.py` implements each engine as a simulator of the *documented* behavior in Table III. Nothing in this subsection is a measurement of a real device. The probe pool has $d = 1$, $N = 4$, policies with one or two rules, priorities in $\{0, 1, 2\}$, quick flag, both defaults, and all flows: 116,160 probes.

The class: $\mathcal{H}_8$ is irredundant on the pool (eight observational classes; 26 distinct response vectors). Exhaustive search finds a minimum distinguishing suite of $3 = \lceil \log_2 8 \rceil$ probes, so the lower bound of Theorem 2(a) is attained again. Algorithm 1 returns 6 probes, within the bound $k - 1 = 7$ and above the minimum, so the greedy suite is not minimum.

TABLE III

DOCUMENTED BEHAVIOR OF FOUR ENGINES AND THE MODELED SEMANTICS. ALL OF THEM HAVE FEATURES OUTSIDE THE MODEL (STATE, USER CHAINS, DEFAULT RULES, ADMINISTRATOR OVERRIDES); ASSUMPTION 4 CONFINES THE RESULTS TO THE FRAGMENT IN THE LAST COLUMN.

Engine	Documented conflict resolution	Model	Fragment (modelability)
iptables [35]	Rules in a chain are examined in order; a matching rule's target (ACCEPT, DROP) decides; at the end of a built-in chain the chain policy applies.	FM, $\delta =$ chain policy	Flat built-in chain with terminal targets only; no user chains or non-terminating targets
pf [36]	Rules evaluated first to last; the last matching block or pass rule decides; a rule with <code>quick</code> is the last matching rule and ends evaluation; default is pass.	LMQ, $\delta =$ permit	block and pass rules without state; match rules excluded
AWS network ACL [37]	Rules evaluated starting with the lowest number; the first match applies regardless of higher-numbered rules.	$PR^\downarrow$ (priority = rule number)	Unique rule numbers; the default deny entry is δ
Azure NSG [38]	Priority 100 to 4096; lower numbers processed first; processing stops at the first match.	$PR^\downarrow$	Custom rules only; security admin rules and default rules excluded

TABLE IV

IDENTIFICATION OF SIMULATED ENGINES WITH $\mathcal{H}_8$. "GREEDY SUITE" IS THE SIZE OF ALGORITHM 1'S SUITE SEPARATING THE OBSERVATIONAL CLASSES OF THE ENGINE'S EXPRESSIBLE POOL.

Engine (simulated)	Expressible probes	Greedy suite	Consistent set
iptables	3360	3	$\{FM, PR^\uparrow, PR^\downarrow\}$
pf	13120	4	$\{LMQ\}$
AWS NACL	19680	4	$\{PR^\downarrow\}$
Azure NSG	19680	4	$\{PR^\downarrow\}$

Engines: Each engine can express only some probes (for example, NACL and NSG priorities are unique, and neither has a quick flag). Restricting the pool accordingly and grouping $\mathcal{H}_8$ into observational classes (Table IV), pf, NACL and NSG are identified uniquely. For iptables, FM, $PR^\uparrow$ and $PR^\downarrow$ agree on every expressible policy because iptables has no priorities, so the consistent set is $\{FM, PR^\uparrow, PR^\downarrow\}$ and Proposition 1 applies. For an allow-only engine all eight hypotheses agree on every expressible policy; no identification is possible and none is needed for decisions on that fragment.

A documented feature outside the class: Take the smaller class $\mathcal{H}_4 = \{FM, LM, PR^\uparrow, BM\}$ with no quick flag, and the suite of Table X. Run against the pf simulator, the consistent set is $\{LM\}$: the suite identifies pf as last-match, because none of its probes uses `quick`. Let the production policy be one quick deny rule on flow 1 with default pass, and let the request be to permit $M = [0, 2]$. A planner using FM inserts the new rule at position 0, and one using LM at the end; each predicts Q1. Evaluated on the pf simulator, both plans leave flow 1 denied, because the quick rule ends evaluation first. With LMQ in the class and quick flags in the pool, the pf simulator is identified as LMQ, the planner inserts a quick permit rule at position 0, and Q1 holds on the simulator. This illustrates Theorem 3: a documented feature missing from the class produced a wrong identification, and widening the class removed the error. The class should therefore be built from the documentation of the engines at hand. Section VII-F repeats the plans on real pf.

An out-of-class engine: An engine that resolves priority ties to the *later* rule is not in $\mathcal{H}_8$. The consistent set of the 6-probe greedy suite is empty, so the deviation is detected. Its uniform distance to the nearest member over the two-rule pool is 0.043 (LM). Detection here depended on the suite containing

tied-priority probes; Theorem 3 says no finite deterministic suite guarantees it.

D. Identification on real kernel engines

The probe oracle in this subsection is the Linux netfilter implementation (kernel 6.18, iptables 1.8.10, nftables 1.0.9), not a simulator. A policy is installed on the input hook of one network namespace, one UDP datagram per flow is sent from a second namespace over a virtual link, and a flow is permitted iff the datagram reaches a bound socket. The scripts are `real_engine_experiment.py` (one dimension, $N = 4$ flows) and `real_engine_experiment2.py` (the engines of Table V and the two-dimensional grid of Table VI). The class $\mathcal{H}_8$ and the suites are those of Section VII-C; only the oracle differs. Before each sweep, controls (permit-all, deny-all, one denied flow) passed, and re-probing a random subset of policies in shuffled order gave no mismatches.

In-class engines: The pool is all one- and two-rule policies with both defaults, probed exhaustively (840 policies, 3360 probes). iptables-legacy, iptables on the `nf_tables` back end, and native nftables with one base chain agree with first-match on every probe, and the three implementations agree with one another on every probe. The consistent set is the observational class $\{FM, PR^\uparrow, PR^\downarrow\}$ that the simulation predicted, and the three-probe greedy suite returns exactly that class. Two further in-class engines behave identically: iptables with the policy in a user chain reached by a jump, and iptables with a second table that only accepts one flow.

Engines outside the class: We built four engines that deviate from $\mathcal{H}_8$ (Table V). The kernel evaluates every base chain registered at a hook, so an accept in one chain does not prevent a drop in another; the measurements below show this. Two engines split the rule list over two base chains of different priority (first rule and rest, first two rules and rest). Two keep the whole policy in iptables and add a second table, as when two tools manage one host, that drops one flow or two flows. On all four, the three-probe identification suite returns a non-empty consistent set. On the engine whose second table drops flow 3 it returns $\{FM, PR^\uparrow, PR^\downarrow\}$, so the engine looks in-class to the suite although that flow is never permitted. These engines show Theorem 3 at work: identification does not certify class membership.

A structured suite of 144 probes flags all four engines and raises no alarm on the three in-class ones. It has one probe for each interaction type of two rules (relation of the two boxes, both actions, both defaults, and which boxes contain the flow) and per-flow coverage under both defaults. We designed it after seeing the first deviation, so this comparison is not blind, and the suite is no smaller than the random suite that already suffices here: $n = 56$ random probes miss every deviation with probability at most 0.0063 by Theorem 4 (the bound $k(1-\varepsilon)^n$ with $k = 8$ and $\varepsilon \geq 0.12$); the measured miss rates in Table V are at most 0.001. For the two-chain engine (first rule and rest), $\varepsilon = 0.120$ with nearest member DO. The empirical miss rate of $n = 1, 2, 3, 5, 8, 12$ random probes (20000 draws) was 0.880, 0.773, 0.679, 0.524, 0.361, 0.212, against the lower bound $(1-\varepsilon)^n = 0.880, 0.774, 0.681, 0.527, 0.359, 0.215$ of Theorem 4 (Monte Carlo standard error at most 0.0035), and $n \geq 56$ suffices for $\delta = 0.01$ by Corollary 1. A plan chosen under FM (insert a permit for flows $[0, 2]$ before an existing deny of flow 1) satisfies Q1 on the three in-class engines and fails on the two-chain engine, where flow 1 stays dropped. In practice, identification should be followed by a separate detection phase before any claim that depends on class membership.

Two dimensions, three to five rules: Flows are pairs (destination port, source address) on a 5×3 grid (15 flows), a rule box is a port range times a source-address range, and the pool is 500 random policies of three to five rules (7500 probes per engine). Priorities and the quick flag are not expressible, so the hypotheses are FM, LM, BM, PO, DO. The three in-class engines agree with FM on all 7500 probes, while the other hypotheses disagree on 9.7% (LM), 4.3% (BM), 5.6% (PO) and 5.3% (DO) of them. The decision is constant on every elementary cell for all 500 policies on every engine, including the two-chain engine, as Lemma 4 predicts since cells depend on rule membership only; the mean is 12.0 cells for 15 flows, so on this small grid cell compression saves only about a fifth of the probes.

For non-interference (Theorem 6) we inserted a permit rule at a random position of a random production policy of two to four rules, in two variants of 200 trials each: the rule box equals the requested match M (exact), or strictly contains it by one cell (loose). On the in-class engines, no exact insertion changed a flow outside M , a loose insertion did in 85 of 200 trials, and the real change set equalled the first-match prediction in all 400 trials. On the two-chain engine, an exact insertion changed a flow outside M in 5 of 200 trials where the model predicts none, and the first-match model mispredicted whether Q1 holds in 146 of 400 trials. Non-interference by exact fidelity is therefore conditional on class membership, as the calculus records.

These runs cover Linux netfilter only, stateless UDP on one hook, at most 15 flows and at most five rules. The two-chain engines are constructions and the second-table engines model coexisting tools on one hook; neither is a claim about deployed configurations. Connection tracking is measured separately in Section VII-G. NAT, zones and other vendors were not measured.

E. A second engine family: Linux routing and nginx

To test whether the pipeline is specific to firewalls, we ran it on two other families with the same class $\mathcal{H}_8$, the same ev , Algorithm 1 and the same exhaustive-sweep protocol (script `real_engine_experiment3.py`). Only the front end and the pool of expressible rules change, as U_e already did for iptables. Routing: flow f is destination $10.99.0.(16+f)$, a box must be an aligned prefix (seven boxes), permit is arrival at a receiver in a second namespace, deny is a blackhole route or rule; three engines: `ip route` (longest prefix, ties by metric = rule index), the same with ties reversed (a deliberate out-of-class control), and `ip rule` (priority = rule index). `nginx 1.24`: flow f is the URI $/b_1b_0$, a prefix location is an aligned box, permit is `return 200`, deny is `return 403`, default permit or deny is a present or absent static file; two `nginx` pools: prefix locations only, and prefix plus exact (`location =`), where an exact location is a single-flow rule with $q = 1$. `nginx` rejects two locations with identical modifier and path, so such policies are excluded from the pool (checked with `nginx -t`). Controls passed and re-probing 25% of policies gave no mismatches on every engine.

Table VII shows three things. In-class engines are identified exactly with a three-probe suite on both new families, and the observational-class structure matches the prediction. The blind-target phenomenon recurs: the reversed-tie route table and `nginx` with exact locations are each accepted by the identification suite and refuted only by the exhaustive sweep, the `nginx` deviation being small ($\varepsilon = .0043$). Finally, `nginx` with exact locations is outside $\mathcal{H}_8$: its rule is exact match first, then longest prefix. Adding that single hypothesis (QBM) to the class and re-running the unchanged procedure identifies it exactly. So the method transfers to these families, but the class did not transfer without change.

Detection phase: We applied the structured suite of Section VII-D to the new engines with its construction rule unchanged: one probe per interaction type of two rules (box relation, both actions, default, containment of the flow) plus per-flow coverage, built from each pool (script `real_engine_experiment4.py`). A suite detects an engine when no hypothesis of the class agrees with it on every suite probe. Table VIII gives the result. The suite raised no alarm on the in-class engines (`ip route`, `ip rule`, `nginx` with prefix locations) nor on `nginx` with exact locations against the extended class $\mathcal{H}_8 \cup \{\text{QBM}\}$, and it detected both out-of-class engines that the identification suite accepted. At $\varepsilon = .0043$ random probing needs about 1,200 probes for $\delta = .05$ (Theorem 4); the structured suite used 112.

We designed the suite after seeing the netfilter deviations, so this is not a blind test; it shows only that the same construction carries over to two other families without redesign (a variant that adds the exact-match flag to the interaction type, 256 probes, also detects the `nginx` deviation and is needed only if the pool is richer). Both deviations are also two-rule interactions, which is the level at which the suite is built. A deviation that depends on three rules or on values inside a box would not be covered, and Theorem 3 applies to this suite as to any finite one.

TABLE V

DETECTION ON REAL NETFILTER ENGINES (ONE DIMENSION, 3360 PROBES, EXHAUSTIVE). $\varepsilon = \min_h d_h$. “3-PROBE” IS THE GREEDY IDENTIFICATION SUITE, “144-PROBE” THE STRUCTURED SUITE, AND THE LAST COLUMN THE MONTE CARLO MISS PROBABILITY OF 56 RANDOM PROBES (N/A WHEN $\varepsilon = 0$).

Engine	ε	3-probe suite	144-probe suite	random $n=56$
iptables (nf_tables back end)	0	in class	no alarm	n/a
iptables, policy in a user chain	0	in class	no alarm	n/a
iptables + second table accepting flow 2	0	in class	no alarm	n/a
iptables + second table dropping flow 3	.125	accepts $\{FM, PR^\uparrow, PR^\downarrow\}$	detects	.001
iptables + second table dropping flows 0–1	.250	accepts $\{DO\}$	detects	.000
nftables, two chains (first rule / rest)	.120	accepts $\{DO\}$	detects	.001
nftables, two chains (first two rules / rest)	.182	accepts $\{DO\}$	detects	.000

TABLE VI

TWO-DIMENSIONAL GRID, 500 POLICIES OF 3–5 RULES, REAL NETFILTER ENGINES. Q2 COLUMNS: TRIALS (OF 200) IN WHICH A FLOW OUTSIDE THE REQUESTED MATCH CHANGED, REAL ENGINE / FIRST-MATCH MODEL. THE LAST TWO COLUMNS COUNT TRIALS (OF 400) IN WHICH THE REAL CHANGE SET, RESPECTIVELY THE REAL TRUTH OF Q1, EQUALS THE FIRST-MATCH MODEL’S.

Engine	d_{FM}	Q2 exact	Q2 loose	Δ equals model	Q1 equals model
iptables-legacy, iptables (nf_tables), nftables one chain (each)	0	0 / 0	85 / 85	400	400
nftables, two chains (first rule / rest)	.157	5 / 0	31 / 85	205	254

TABLE VII

SECOND FAMILY, EXHAUSTIVE SWEEPS. $|U_e|$ IS THE NUMBER OF PROBES. “SUITE” IS THE GREEDY IDENTIFICATION SUITE OVER THE OBSERVATIONAL CLASSES OF THE ENGINE’S POOL.

Engine	$ U_e $	Suite	Result	ε
ip route	1680	3	BM, exact	0
ip rule	1680	3	FM class, exact	0
ip route, rev. ties	1680	3	LM accepted; refuted	.0286
nginx, prefix only	1456	3	BM, exact	0
nginx, prefix+exact	3696	4	BM accepted; refuted	.0043
with QBM added	3696	5	QBM, exact	0

TABLE VIII

DETECTION ON OUT-OF-CLASS ENGINES. “IDENT.” IS THE IDENTIFICATION SUITE (3 AND 4 PROBES), S_{cc} THE STRUCTURED SUITE (112 PROBES). THE LAST TWO COLUMNS GIVE THE MONTE CARLO PROBABILITY (4000 TRIALS) THAT n UNIFORMLY RANDOM PROBES MISS THE DEVIATION.

Engine	ε	Ident.	S_{cc}	$n=56$	$n=200$
ip route, rev. ties	.0286	accepted	detected	.40	.005
nginx, prefix+exact	.0043	accepted	detected	.90	.47

These two further families, one kernel and one server, have at most two rules, four flows and a single dimension. The ties-by-metric convention for `ip route` and the exclusion of regular-expression locations are adapter choices fixed before the runs. Cedar, OPA and cloud IAM were not tested.

F. A second operating system: OpenBSD pf

Table III gives pf’s documented behavior as last-match with a quick override. We checked it on OpenBSD 8.0 (GENERIC, amd64, no syspatches installed), where pf is the system packet filter (script `real_engine_experiment6_pf.py`). A policy is loaded with `pfctl` with the ruleset optimizer off (`-o none`), one UDP datagram per flow is sent to `127.0.0.1` over `100`, and a flow is permitted iff it reaches a bound socket. Rules are inbound UDP rules on `100` without state, so each

probe depends on rule evaluation only. The default permit is pf’s own, and default deny is a leading non-quick block-all rule. An IPv6 datagram to `:::1`, which the IPv4-only rules always pass, is sent last, so a late IPv4 packet is not read as a deny. The controls passed: permit-all, deny-all, one denied flow, a quick deny before a permit, and a plain deny before a permit.

The pool is the one of Section VII-C: 13,120 probes from one- and two-rule policies with the quick flag, both defaults, probed exhaustively. All 13,120 responses agree with LMQ, and each of the other seven hypotheses disagrees on 6.3% of probes. The four-probe greedy suite returns $\{LMQ\}$, the same answer as the simulation, and re-probing 820 policies in shuffled order gave no mismatches. For the plan example of Section VII-C, the two-probe suite for the quick-free class returns $\{LM\}$, plans chosen under FM and under LM both leave flow 1 denied on the real filter, and the plan chosen under LMQ, a quick permit at position 0, permits flows 0 to 2. The wrong plan therefore occurs on a real packet filter as well as in the simulator.

The documented behavior is therefore right and the class identifies it exactly, for loopback traffic, stateless rules, and inbound rules on one interface and one release.

G. When oracle fidelity fails silently: connection tracking

Every engine so far was stateless or a construction of ours, and the sandbox was assumed faithful. Real rule sets usually begin with an accept for established connections (ct state established, related accept), and connection-tracking entries outlive a rule change. We measured what that does to a sandbox (script `real_engine_experiment7_state.py`, kernel 6.18.44, nftables 1.0.9). The setup is the one of Section VII-D, one input chain with the policy, the same 840 one- and two-rule policies, probed in a shuffled order. The receiver echoes each datagram, so the entry becomes established. We ran three modes, each on fresh namespaces.

- *plain*: no connection-tracking rule. Every flow keeps one source port.
- *fresh*: the accept rule is present and every probe uses a new source port.
- *stale*: the accept rule is present and every flow keeps one source port, as a sandbox that reuses its sockets does.

Plain and fresh are in class: the controls pass, $\varepsilon = 0$, the three-probe suite returns $\{\text{FM}, \text{PR}^\uparrow, \text{PR}^\downarrow\}$, and re-probing 40 policies gave no mismatch. In the stale mode the engine permitted 1,678 of 3,360 probes that the policy denies, and denied none that it permits. Its disagreement with the members of $\mathcal{H}_8$ is between .4375 and .5613, so $\varepsilon = .4375$, and the identification suite returns $\{\text{PO}\}$, a wrong answer. The cause is that an entry created while a flow was permitted keeps accepting it under every later policy, so after a short warm-up the engine behaves like permit-all on the flows seen before.

The plan check of Section VI’s worked request shows the consequence. We probed permit-all first, installed P_0 (deny flow 1, default deny) and applied two plans. On the fresh sandbox the plan chosen under last-match is correctly rejected: the engine returns (1, 0, 1, 0) and flow 1 stays denied. On the stale sandbox the engine returns (1, 1, 1, 1) for both plans. The check then reports Q1 as holding for the wrong plan, and Q2 fails for flow 3, which the policy denies. A model-based check and an apply-and-diff check are fooled in the same way, because both rest on the sandbox answering as a function of the policy and the flow. This is a failure of Assumption 1, not of the class. It is a property of how the sandbox is driven rather than of the rule engine, and that is the point: nothing in the identification procedure reveals it. Replaying probes does not help here, since the stale engine answers the same way twice (0 mismatches in 160 re-probed responses). The control run does: permit-all followed by deny-all returns (1, 1, 1, 1) on the stale sandbox, and a failed control is a cheap guard to run before any identification. Fresh source ports, or flushing connection-tracking state between probes, remove the effect.

The result is one engine and one idiom on one kernel, and a model that omits state predicts the effect. What it adds is its size, its invisibility to the identification suite, and the cheap guard.

H. Deviations drawn by a fixed generator

The structured suite of Sections VII-D and VII-E was designed after we had seen deviations, so its success says little about engines it was not built around. We therefore ran a second test (script `real_engine_experiment5.py`) in which the deviating engines come from a seeded generator written before any engine was run. The suites are the ones already defined: the identification suite S_{id} (3 probes) and the structured suite S_{cc} (144 probes, built from policies of at most two rules by the unchanged construction rule). We added one suite, S_3 (144 probes), which applies the same idea to three-rule policies: one probe for each pattern of the three actions, the default and which rules contain the flow. It was fixed before the run, and its results are reported whatever they are.

Every engine is real kernel behavior, installed with one `nft` call per policy. The generator draws each engine from three

TABLE IX
ENGINES FROM THE FIXED GENERATOR, THREE SEEDS POOLED. ENTRIES ARE ENGINES DETECTED OUT OF ENGINES IN THE GROUP; ε IS THE ESTIMATED DISTANCE TO $\mathcal{H}_8$ UNDER μ . THE LAST TWO COLUMNS GIVE THE RANGE, OVER THE ENGINES OF THE GROUP, OF THE PROBABILITY THAT n RANDOM PROBES MISS THE DEVIATION.

Group	ε	S_{id}	S_{cc}	S_3	$n=56$	$n=200$
in class (decoys)	0	0/6	0/6	0/6	n/a	n/a
always drop	.123–.125	0/31	31/31	31/31	0–.001	0
late split, $k=1$	.086	0/17	17/17	17/17	.007	0
late drop, $m=2$	.082–.083	0/26	26/26	12/26	.008–.010	0
late split, $k=2$	.068	0/18	0/18	18/18	.037	0
late drop, $m=3$	.040–.043	0/28	0/28	8/28	.088–.108	0
late drop, $m=4$	0 under μ	0/3	0/3	0/3	1	1

families. A *late split* at k sends the rules after the first k to a second base chain, but only when the policy has more than k rules ($k \in \{1, 2\}$). A *late drop* adds a second table that drops one flow, but only when the policy has at least m rules ($m \in \{2, 3\}$). An *always drop* adds such a table for every policy. Each run also contains two in-class decoys, one plain and one with a second table that accepts a flow, and one engine that drops a flow only for policies of four or more rules. Engines are judged on a distribution μ with the rule count uniform on $\{1, 2, 3\}$, then a uniform policy and a uniform flow, sampled with 600 policies per count (40 for one rule, which is all there are). The distance ε is the smallest disagreement with any member of $\mathcal{H}_8$ on that sample, so it is an estimate. Random probing is simulated from the recorded responses (10,000 trials per engine).

We ran three seeds of 40 generated engines each on a second machine (kernel 6.17, `nftables 1.1.5`), which gives 120 generated engines, 6 decoys and 3 four-rule engines.

Table IX shows four things. First, the three-probe identification suite accepts all 123 deviating engines, as Theorem 3 predicts, and the decoys raise no alarm under any suite. Second, S_{cc} detects every engine whose deviation shows on two rules (74) and none of the 46 whose deviation needs three: the late drops with $m = 3$ and the late splits with $k = 2$. This is the limit of a suite built from two-rule interactions. Third, S_3 detects 86 of the 120 generated engines. Its misses follow one rule: among the 54 late drops, it detects exactly those whose dropped flow is 0 or 1 (20 engines) and none of the others (34). Its three-rule probes all use flows 0 and 1, since those were the first representatives in the pool order, and it contains no two-rule probes. A pattern-based suite does not choose which flow it asks about. Fourth, the engines that deviate only at four rules are invisible to every suite and to μ itself ($\varepsilon = 0$ on the sample), yet about 13% of four-rule probes disagree with first-match. Here the deviation lies outside everything the probes touched, as Theorem 3 allows.

Random probing did better than either structured suite at moderate cost. With $\varepsilon \geq .040$, $n = 56$ random probes miss with probability up to .108, while $n = 200$ missed in none of 10,000 trials for any of the 120 generated engines, so the one-sided 95% upper bound on the miss probability is about $3/10,000$ per engine [39]. This agrees with Corollary 1, which asks for $n \geq \varepsilon^{-1} \ln(k/\delta) \approx 128$ at $\varepsilon = .040$, $k = 8$, $\delta = .05$.

Structured suites pay off when ε is very small, as for nginx with exact locations ($\varepsilon = .0043$), and not when deviations touch several percent of probes.

The test has limits. The generator and its three families are ours, so the test is blind with respect to the suites and the parameters, not with respect to the kinds of deviation. The engines are constructions on one hook, the policies have at most four rules and four flows, the two machines differ in kernel version but not in kind, and the ε values rest on samples of 1240 policies per engine.

VIII. LIMITATIONS

- 1) *Model gap*. Real engines have zones, translation, state and application identification. Assumptions 2–4 confine the results to policies the model expresses. Outside that fragment, verification is partial.
- 2) *Declared, not discovered*. $\mathcal{H}$ is chosen by the designer. The results are conditional on the target being in $\mathcal{H}$; Theorem 3 shows this cannot be removed by finite probing. How well a given $\mathcal{H}$ covers real vendors is an empirical question not addressed here.
- 3) *Sandbox fidelity*. Assumption 1 requires a faithful sandbox, which many deployments lack.
- 4) *Known ingredients*. The identification, cell and differencing results are instances of known work (Section II). The contribution claimed is the blind-target statement, the measurements, and the composition with its stated conditions. The survey of related work is not exhaustive.
- 5) *Detection bounds*. Theorem 4 is for oblivious targets and i.i.d. probes. Targeted probe design is not analyzed.
- 6) *Complexity*. The cell bound is exponential in d . Practical implementations would use decision diagrams; their analysis is omitted.
- 7) *Not covered*. Concurrency and freshness of dependencies, scoped execution against a compromised agent, multi-step change, and data-disclosure properties are outside this paper.
- 8) *Narrow real-device evidence*. Real measurements cover Linux netfilter (UDP on one hook, with connection tracking in one experiment only, at most 15 flows and five rules) and, at smaller scale, Linux routing and nginx, on two kernels, and OpenBSD pf on loopback with stateless rules. AWS and Azure are simulators of documented behavior, which is our reading of the documentation and can be wrong. The engines outside the class are constructions. We designed the structured suite after the first deviation was observed. The generator experiment of Section VII-H tests it on engines it was not built around, but the generator and its families are ours, and deviations that need three or more rules are found only by a suite built for them. In the first netfilter experiments every deviation had $\varepsilon \geq 0.12$, and in the generator experiment ε was at least .040 except for the four-rule engines. The only smaller deviations ($\varepsilon = .0286$ in routing and .0043 in nginx) are single constructions, so detection of very small deviations rests on two cases.

TABLE X
A MINIMUM DISTINGUISHING SUITE FOR {FM, LM, PR, BM} ($d=1$,
DEFAULT deny, FLOW 0). RULES ARE (BOX, ACTION, PRIORITY).

Policy rules	FM	LM	PR	BM
$([0, 0], \text{deny}, 0), ([0, 0], \text{permit}, 1)$	deny	permit	permit	deny
$([0, 1], \text{deny}, 0), ([0, 0], \text{permit}, 0)$	deny	permit	deny	permit

IX. CONCLUSION

Sandbox probes identify the conflict-resolution semantics of an ordered-rule policy engine exactly when they distinguish every pair of candidates, but they cannot show that the engine belongs to the declared class. On real Linux netfilter, routing and nginx engines, a three-probe suite identified every in-class engine and accepted a wrong hypothesis on every engine outside the class that we built or that a seeded generator drew, so detection needs its own phase. A structured suite finds the deviations it is built for and misses those that need more rules than it covers, and random probing is the safer default when deviations touch several percent of probes. Given an identified semantics, permission for the requested flows reduces to a finite check, non-interference is exactly decidable and automatic under exact fidelity, and an assurance calculus ties every established claim to named assumptions. The identification, cell and differencing ingredients are known; the contribution lies in the blind-target statement, the measurements and their composition. On OpenBSD pf the same procedure recovered the documented behavior exactly, and on Linux a rule set with the common established-connections accept made a reused sandbox answer from the history of earlier probes, which a control run detects. The behavior of AWS and Azure is simulated, not measured.

APPENDIX

These checks use a fixed seed (script `sanity_checks.py`) and small instances. They are not an evaluation on real systems.

Minimum suite for the four semantics: Universe: $d = 1$, $N = 4$, policies with one to three rules (boxes are intervals, priorities in $\{0, 1, 2\}$, both defaults), all flows. All 16 response vectors in $\mathbb{D}^4$ over (FM, LM, PR, BM) occur; the class is irredundant. Exhaustive search over subsets of response vectors finds that the minimum distinguishing suite has size $2 = \lceil \log_2 4 \rceil$, so the lower bound of Theorem 2(a) is attained here. Table X gives one such suite.

Cells, locality, differencing: On $d = 2$, $N_j = 6$: for 3000 random policies with one to five rules, every cell of Lemma 4 was decision-constant under all four semantics and the cell count never exceeded $(2n + 1)^d$; the bound was attained at least once (maximum ratio 1.0). For 2000 random insertions (a permit rule with random box, position and priority; requested match M either equal to the rule’s box or a random sub-box), evaluated under each semantics (8000 trials), the set Δ computed from cells equalled the brute-force Δ in every trial and $\Delta \subseteq B$ in every trial (Lemma 3). Q2 held in 6471 trials and failed in 1529; all failures had M strictly inside B , and none had $B = M$, as Theorem 6(i) predicts.

TABLE XI
EMPIRICAL MISS RATE OF RANDOM PROBING AGAINST THE BOUNDS OF
THEOREM 4. $\varepsilon = \min_h d_h$.

ε	n	lower $(1 - \varepsilon)^n$	empirical	upper $\sum_h (1 - d_h)^n$
0.0204	50	0.3564	0.3548	0.4217
0.0204	100	0.1270	0.1308	0.1291
0.0204	200	0.0161	0.0168	0.0161
0.0496	50	0.0785	0.0807	0.0940
0.0496	100	0.0062	0.0048	0.0063
0.0496	200	3.8×10^{-5}	0.0001	3.8×10^{-5}

Detection bounds: Pool: all $d = 1$, $N = 4$ policies with one or two rules, both defaults, all flows, with μ uniform. The target was PR with a random subset of probes flipped (target deviation 0.02 and 0.05). With 20000 random draws per row, the empirical miss rate lies within the bounds of Theorem 4 up to Monte Carlo error (Table XI). The one value below its lower bound (0.0048 against 0.0062) is 2.5 standard errors away, and a second run with another seed and 100,000 draws per row stayed inside the bounds.

STATEMENTS AND DECLARATIONS

Funding. No funding was received for conducting this study.

Competing interests. The author has no relevant financial or non-financial interests to disclose.

Ethics approval, consent to participate and consent for publication. Not applicable. The study involved no human participants or animals.

Author contributions. The sole author conceived the study, developed the theory, wrote the software, ran the experiments, and wrote the manuscript.

Data and code availability. The code, experiment outputs and specification supporting this study are available at <https://github.com/honbles/outcome-Integration>.

REFERENCES

- [1] A. Clemm, L. Ciavaglia, L. Z. Granville, and J. Tantsura, "Intent-based networking - concepts and definitions," Internet Research Task Force, RFC 9315, 2022.
- [2] S. Jha, S. Gulwani, S. A. Seshia, and A. Tiwari, "Oracle-guided component-based program synthesis," in *Proc. ACM/IEEE International Conference on Software Engineering (ICSE)*, 2010, pp. 215–224.
- [3] E. F. Moore, "Gedanken-experiments on sequential machines," in *Automata Studies*, ser. Annals of Mathematics Studies, C. E. Shannon and J. McCarthy, Eds. Princeton University Press, 1956, no. 34, pp. 129–153.
- [4] T. S. Chow, "Testing software design modeled by finite-state machines," *IEEE Transactions on Software Engineering*, vol. SE-4, no. 3, pp. 178–187, 1978.
- [5] M. G. Gouda and A. X. Liu, "Structured firewall design," *Computer Networks*, vol. 51, no. 4, pp. 1106–1120, 2007.
- [6] A. X. Liu and M. G. Gouda, "Diverse firewall design," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 9, pp. 1237–1251, 2008.
- [7] A. Das and D. Gupta, "Identifying intelligent processes via online sequential testing," arXiv preprint arXiv:2609.26193, 2026.
- [8] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, "Network configuration protocol (NETCONF)," Internet Engineering Task Force, RFC 6241, 2011.
- [9] M. Bjorklund, "The YANG 1.1 data modeling language," Internet Engineering Task Force, RFC 7950, 2016.
- [10] HashiCorp, "The terraform apply command," Terraform CLI documentation, <https://developer.hashicorp.com/terraform/cli/commands/apply>.

- [11] P. Kazemian, G. Varghese, and N. McKeown, "Header space analysis: Static checking for networks," in *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2012, pp. 113–126.
- [12] A. Fogel, S. Fung, L. Pedrosa, M. Walraed-Sullivan, R. Govindan, R. Mahajan, and T. Millstein, "A general approach to network configuration analysis," in *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2015, pp. 469–483.
- [13] J. Backes, P. Bolignano, B. Cook, C. Dodge, A. Gacek, K. Luckow, N. Rungta, O. Tkachuk, and C. Varming, "Semantic-based automated reasoning for AWS access policies using SMT," in *Proc. Formal Methods in Computer-Aided Design (FMCAD)*, 2018, pp. 206–214.
- [14] E. Al-Shaer and H. Hamed, "Discovery of policy anomalies in distributed firewalls," in *Proc. IEEE INFOCOM*, vol. 4, 2004, pp. 2605–2616.
- [15] R. Birkner, T. Brodmann, P. Tsankov, L. Vanbever, and M. Vechev, "Metha: Network verifiers need to be correct too!" in *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2021, pp. 99–113.
- [16] A. El-Atawy, T. Samak, Z. Wali, E. Al-Shaer, F. Lin, C. Pham, and S. Li, "An automated framework for validating firewall policy enforcement," in *Proc. IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY)*, 2007, pp. 151–160.
- [17] T. Samak, A. El-Atawy, and E. Al-Shaer, "Firewall policy reconstruction by active probing," in *Proc. 2nd Workshop on Secure Network Protocols (NPsec)*, 2006.
- [18] T. Samak, A. El-Atawy, and E. Al-Shaer, "FireCracker: A framework for inferring firewall policies using smart probing," in *Proc. IEEE International Conference on Network Protocols (ICNP)*, 2007, pp. 294–303.
- [19] M. K. Alhamwi, O. Al-Hmouz, M. H. Sqalli, and K. Salah, "A delay-based probing technique for the discovery of a firewall's accept rules," in *Proc. IEEE GCC Conference and Exhibition*, 2011, pp. 445–448.
- [20] K. Salah, K. Sattar, M. Sqalli, and E. Al-Shaer, "A probing technique for discovering last-matching rules of a network firewall," in *Proc. International Conference on Innovations in Information Technology (IIT)*, 2008, pp. 578–582.
- [21] G. Argyros, I. Stais, A. Kiayias, and A. D. Keromytis, "Back in black: Towards formal, black box analysis of sanitizers and filters," in *Proc. IEEE Symposium on Security and Privacy*, 2016, pp. 91–109.
- [22] G. Argyros, I. Stais, S. Jana, A. D. Keromytis, and A. Kiayias, "SFADiff: Automated evasion attacks and fingerprinting using black-box differential automata learning," in *Proc. ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2016, pp. 1690–1701.
- [23] P. Iyer and A. Masoumzadeh, "Learning relationship-based access control policies from black-box systems," *ACM Trans. Privacy and Security*, vol. 25, no. 3, 2022.
- [24] J. W. Cutler, C. Disselkoen, A. Eline, S. He, K. Headley, M. Hicks, K. Hietala, E. Ioannidis, J. Kastner, A. Mamat, D. McAdams, M. McCutchen, N. Rungta, E. Torlak, and A. Wells, "Cedar: A new language for expressive, fast, safe, and analyzable authorization," *Proc. ACM Program. Lang. (OOPSLA)*, 2024, arXiv:2403.04651.
- [25] C. Disselkoen, A. Eline, S. He, K. Headley, M. Hicks, K. Hietala, J. Kastner, A. Mamat, M. McCutchen, N. Rungta, B. Shah, E. Torlak, and A. Wells, "How we built Cedar: A verification-guided approach," in *Companion Proc. 32nd ACM Int. Conf. on the Foundations of Software Engineering (FSE Companion)*, 2024.
- [26] C. D. P. K. Ramli, H. R. Nielson, and F. Nielson, "The logic of XACML," 2011, arXiv:1110.3706.
- [27] D. Xu, Y. Zhang, and N. Shen, "Formalizing semantic differences between combining algorithms in XACML 3.0 policies," in *Proc. IEEE Int. Conf. on Software Quality, Reliability and Security (QRS)*, 2015, pp. 163–172.
- [28] K. Fislir, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz, "Verification and change-impact analysis of access-control policies," in *Proc. 27th Int. Conf. on Software Engineering (ICSE)*, 2005.
- [29] H. Liu, Y. Zhu, J. Padhye, J. Cao, S. Tallapragada, N. Lopes *et al.*, "CrystalNet: Faithfully emulating large production networks," in *Proc. 26th Symposium on Operating Systems Principles (SOSP)*, 2017.
- [30] A. Krentsel, O. Ye, A. Tafoya, X. Ma, S. Ratnasamy, and A. Shaikh, "Towards accessible model-free verification," in *Proc. 24th ACM Workshop on Hot Topics in Networks (HotNets)*, 2025.
- [31] S.-J. Moon, J. Helt, Y. Yuan, Y. Bieri, S. Banerjee, V. Sekar, W. Wu, M. Yannakakis, and Y. Zhang, "Alembic: Automated model inference for stateful network functions," in *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2019, pp. 699–718.
- [32] D. Angluin, "Learning regular sets from queries and counterexamples," *Information and Computation*, vol. 75, no. 2, pp. 87–106, 1987.

- [33] G. C. Necula, “Proof-carrying code,” in *Proc. ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, 1997, pp. 106–119.
- [34] A. Pnueli, M. Siegel, and E. Singerman, “Translation validation,” in *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, ser. LNCS, vol. 1384, 1998, pp. 151–166.
- [35] Netfilter project, “iptables(8),” Linux manual page, <https://man7.org/linux/man-pages/man8/iptables.8.html>.
- [36] OpenBSD, “pf.conf(5): packet filter configuration file,” Manual page, <https://man.openbsd.org/pf.conf.5>.
- [37] Amazon Web Services, “Network ACL rules,” Amazon VPC User Guide, <https://docs.aws.amazon.com/vpc/latest/userguide/nacl-rules.html>.
- [38] Microsoft, “Network security groups overview,” Azure documentation, <https://learn.microsoft.com/en-us/azure/virtual-network/network-security-groups-overview>.
- [39] J. A. Hanley and A. Lippman-Hand, “If nothing goes wrong, is everything all right? Interpreting zero numerators,” *JAMA*, vol. 249, no. 13, pp. 1743–1745, 1983.