An Efficient User–Kernel Access Control Scheme for Lightweight Data Connectors

Mengxiang Zhu[1,2,3], Wenlong Kou✉[1,3] ⋆, Yunchuan Guo[1,2,3], Yanru He[1,3], Kui Geng[1,2,3], and Liang Fang[1,2,3]

[1] Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
[3] State Key Laboratory of Cyberspace Security Defense, Beijing, China
{zhumengxiang,guoyunchuan}@iie.ac.cn

Abstract. Data connectors are key components for enabling controllable data exchange in data marketplaces and constitute one of the foundational technologies for realizing data sovereignty. To support controllable data exchange in lightweight data connectors, it is crucial to design an efficient and scalable access control mechanism. However, user-space access control suffers from high decision latency, whereas kernel-space access control is limited by constrained policy capacity and inflexible policy updates. To address these challenges, we propose an efficient user–kernel access control scheme for lightweight data connectors. The scheme leverages extended Berkeley Packet Filter (eBPF) to load high-frequency access control rules into the kernel while retaining low-frequency rules in user space. To further improve access control efficiency, we design a Bloom-filter-based kernel admission filter to block invalid requests before they enter the policy decision point (PDP). To prevent long-tail requests from polluting the kernel cache, we develop a set-associative, access-frequency-aware cache replacement mechanism. To determine the optimal memory allocation between these two kernel-side components, we model access control decisions as an M/D/1 queueing system, and analyze the relationship between end-to-end policy decision latency and cache miss probability to guide the memory partitioning between the two kernel-side components. We implement a prototype based on eBPF. Experimental results show that, under the same policy set size, our system reduces the average service time by approximately 25% compared with AppArmor; with 10^6 policies, it reduces the average service time by up to 85.6% compared with Casbin. Under mixed invalid and long-tail traffic with an attack ratio $\gamma > 80\%$, the proposed design reduces latency by up to 65% compared with an unprotected LRU policy.

Keywords: Data connectors · eBPF · Access control · Cache replacement.

1 Introduction

International Data Spaces (IDS) is an architectural framework that enables trusted data exchange and data-driven services among heterogeneous stakeholders, and

⋆ Corresponding author: kouwenlong@iie.ac.cn

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_27

it is emerging as a secure and standardized digital infrastructure for data trading. Within IDS, the data connector is a core component that enforces data usage policies to ensure that data is accessed and processed strictly under the conditions specified by the data provider. In practice, a data connector may be deployed either on end systems or resource-constrained edge devices close to data sources. In this scenario, the data connector is required to be lightweight. Since every access request must be mediated by the access control module embedded in the data connector, it is essential to reduce policy-enforcement overhead to achieve efficient access control for lightweight IDS data connectors.

From the perspective of the deployment location of access control, existing access control schemes can be grouped into three categories: user-space access control [20], kernel-space access control [3], and hybrid access control [1]. In user-space access control, both the policy decision point (PDP) and the policy enforcement point (PEP) are deployed at the application layer, which simplifies deployment and facilitates policy upgrades. However, the scheme introduces non-negligible latency, making it difficult to satisfy the stringent low-latency requirements of high-frequency data access scenarios. In contrast, kernel-space access control colocates the PDP and PEP within the kernel, achieving higher execution efficiency; yet, due to limited kernel resources, it significantly complicates policy updates. Hybrid access control aims to reconcile these trade-offs by caching frequently matched rules in the kernel and dynamically evicting cold entries, thereby enabling fast decision making and enforcement, while keeping low-frequency rules in user space to mitigate kernel resource constraints. Nevertheless, when applied to data connectors, hybrid access control still exhibits the following limitations and challenges:

1. **Long-tail cold data pollution.** Long-tail cold data originates from two sources: malicious access requests and long-tail request data generated by normal business operations. An adversary can deliberately craft malicious requests with long-tail characteristics to evict frequently matched rules from the cache, thereby substantially reducing the cache hit rate. Once a significant fraction of genuine high-frequency rules is displaced, the end-to-end decision latency increases sharply, leading to degraded quality of service.
2. **Kernel cache size optimization.** To filter out malicious access requests, detection code must be executed within the kernel space. In general, the more memory allocated to the malicious request detection module, the higher the real-time detection rate [6]. However, increasing the memory for detection directly reduces the memory available for caching high-frequency rules in the kernel, thereby decreasing the cache hit rate and vice versa. Consequently, determining an optimal kernel memory partition that balances detection accuracy against cache-hit performance constitutes a key challenge.

These two limitations and challenges significantly impair the efficiency of access-control decision making. To address the above challenges, we propose UKAC, a lightweight user–kernel hybrid access control scheme for data connectors. Our main contributions are as follows:

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_27

- **Anti-pollution cache replacement mechanism:** We employ a Bloom filter to intercept malicious access requests targeting non-existent objects. Moreover, to comply with the eBPF verifier's bounded-loop constraint, we partition cached rules into hash-indexed groups. When the hit count of a rule exceeds a predefined threshold, UKAC locates the corresponding group using the rule's hash value and evicts the least frequently used entry within that group, thereby mitigating cache pollution caused by long-tail cold traffic.
- **Queueing-theoretic kernel cache sizing:** To accurately configure the kernel cache size, we model the user–kernel interaction as an M/D/1 queueing system and derive the relationship between the target expected latency $E[T]$ and cache size, providing a quantitative basis for cache provisioning.
- **Prototype implementation and evaluation:** We implement a prototype of UKAC for lightweight data connectors. Experimental results show that, under the same policy scale, UKAC reduces the mean service time by approximately 25% compared with AppArmor, and by up to 85.6% compared with Casbin at 10^6 policies. Under mixed invalid and long-tail traffic with attack ratio $\gamma > 80\%$, UKAC reduces $E[T]$ by up to 65% relative to an unprotected LRU baseline.

Availability. The source code of the UKAC prototype is publicly available at `https://github.com/ukac-paper/ukac`.

2 Related Work

2.1 Access Control

In data connectors, access control policy enforcement requires balancing policy execution efficiency against policy expressiveness. As noted above, existing access control schemes can be grouped into three categories: user-space access control, kernel-space access control, and hybrid access control. These categories differ in where access requests are intercepted and evaluated, which in turn affects end-to-end latency, throughput, and scalability.

In user-space access control systems, the policy decision point (PDP) and the policy enforcement point (PEP) are either embedded within the application itself [23] or deployed at a service gateway [7]. To enable fine-grained authorization, typical models include role-based access control (RBAC) [11] and attribute-based access control (ABAC) [31,17]. For example, Amazon Web Services (AWS) realizes ABAC by encoding attributes as tags and enforcing tag matching or tag constraints in IAM policies. Eiers et al. [13] have further performed quantitative analyses to characterize and reason about policy permissiveness at scale. In online social networks (OSNs), relationship-based access control is used to reduce authorization to checking whether the requester and the target object (or resource owner) are connected by a relationship path that satisfies a specified predicate, thereby providing expressive visibility control [9]. Wang et al. abstract user-facing access-control requirements into three dimensions: (i) users should be able to inspect/query access-control state (e.g., controllers, fabrics, and device-side

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_27

ACLs), (ii) the presented access-control information should be trustworthy, and (iii) users should be able to update/revoke permissions to mitigate the risk of loss of control over IoT devices [26].

To improve policy flexibility and scalability, Luo et al. [19,20] decouple cloud access control into an independent user-space policy engine (or authorization service) and employ a sidecar proxy as an external PEP to intercept requests and forward them for decision making. However, under high-frequency data-exchange workloads, such a purely user-space design increases the number of user–kernel boundary crossings and memory copies, which in turn inflates end-to-end decision latency and degrades system throughput.

Kernel-space access control, in contrast, embeds enforcement logic directly into the operating-system kernel and makes authorization decisions using access control lists (ACLs) [15], capability-based models [16], or mandatory access control (MAC) mechanisms [25]. SELinux [18] and AppArmor [3] provide general-purpose frameworks for integrating such mechanisms into the kernel. Recent work has further extended kernel-side enforcement to more dynamic settings, including context-driven permission transitions via finite state machines [8], fine-grained privilege delegation [27], and system-call interception and isolation for containers [12,28]. Nevertheless, these approaches are difficult to apply directly in resource-constrained deployment environments for lightweight data connectors: the kernel execution environment is subject to strict resource and safety constraints, making it impractical to load, update, and evaluate large and rapidly evolving policy sets without jeopardizing system stability. This limitation prevents such approaches from meeting the requirements of IDS data connectors in terms of both policy expressiveness and dynamic updates.

Hybrid access-control architectures aim to balance kernel-side execution efficiency with the expressiveness of user-space policy evaluation. Along this line, Amitu et al. and Thakare et al. [4,24] adopt hybrid or hierarchical designs in IoT and highly concurrent scenarios, e.g., by combining multiple access mechanisms or extending authorization models to handle concurrency conflicts and policy-management issues. However, these efforts primarily focus on protocol-level or cloud-side authorization management, and do not address the operating-system-level constraints encountered by user–kernel collaborative enforcement.

2.2 Cache Replacement Algorithms

Classic cache replacement policies can be divided into two categories: recency-based Least Recently Used (LRU) and frequency-based Least Frequently Used (LFU). LRU always evicts the least recently accessed object [30], offering straightforward implementation and effective performance under workloads with strong temporal locality. LFU performs replacement based on access frequency and has been shown to achieve optimal hit rates under the Independent Reference Model (IRM) [2]. Researchers have further proposed a variety of algorithms that combine recency and frequency [29,14,5], among which Adaptive Replacement Cache (ARC) [22] and TinyLFU [14] are representative. ARC simultaneously

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_27

maintains two categories of objects–recently accessed and frequently accessed–along with their ghost histories, and dynamically adjusts the proportion between them to adapt to changing access patterns, thereby evicting stale objects while retaining current hotspots with $O(1)$ overhead. TinyLFU leverages approximate data structures such as Bloom filters and Count-Min sketches [10] to record historical access frequencies, and employs an admission control mechanism that retains the higher-frequency item between a candidate newcomer and the eviction candidate, reducing decision complexity to $O(1)$ and thus caching hot objects more efficiently.

Although these algorithms perform well under various conditions, they typically rely on complex data structures and extensive state maintenance. The eBPF verifier prohibits unbounded loops and excessively large global state, which renders the standard TinyLFU algorithm difficult to implement within the eBPF environment.

3 System Modeling

3.1 Observation and Assumptions

Our design is grounded in statistical analysis of real-world data connectors, from which we derive the following system characteristics:

- **Observation: Heavy-Tailed Access Distribution:** We collected real-world access logs from a large enterprise application spanning five years, comprising 62,952 access records in total. After performing a series of statistical analyses on the logs, we observe that access requests are highly skewed toward a small subset of frequently matched rules and follow a Zipfian heavy-tailed (long-tailed) distribution with parameter $\alpha \approx 1$.
- **Assumption 1 (Kernel Memory Constraint):** Let $N = |\mathcal{R}|$ denote the total number of policy rules in the rule set, and let M denote the upper bound on kernel caching capacity, i.e., the maximum number of rule entries that can be stored in kernel space. For lightweight data connectors, we assume a resource-constrained kernel memory budget and thus $N \gg M$.
- **Assumption 2 (Read-Dominated Workload):** From a memory-access perspective, policy operations fall into two categories: policy enforcement (reads) and policy updates (writes). We assume a read-dominated workload where the arrival rate of read requests λ_r is significantly higher than that of write requests λ_u, i.e., $\lambda_r \gg \lambda_u$.

3.2 Threat Model

In resource-constrained data connector deployments, both compute and storage budgets are extremely limited, and any non-essential overhead can directly degrade quality of service (QoS). In such settings, a strategic adversary can exploit the asymmetry between fast kernel-space decisions and comparatively slow user-space processing. By generating carefully crafted access requests, the

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_27

adversary can steer benign requests away from kernel space and force them into user space, thereby reducing the availability of the access-control service. In particular, we consider two primary attack vectors:

- **Invalid access request flooding:** The adversary continuously issues requests targeting non-existent resources or requests for which no applicable policy is defined. Since these requests cannot match any rule cached in the kernel, they are forwarded to user space for evaluation. This flooding attack amplifies per-request processing overhead and consumes compute resources needed by legitimate requests.
- **Long-tail traffic flooding:** By observing response-time differences, the adversary can infer the heavy-tailed access pattern of policy rules and then generate a large volume of valid but extremely low-frequency long-tail requests. Because conventional caching policies are often recency-biased, these long-tail accesses can rapidly occupy scarce kernel memory and evict genuinely high-frequency rules from the cache. As a result, a substantial fraction of benign requests is forced onto the user space, leading to increased end-to-end latency and degraded QoS.

3.3 Architecture Overview

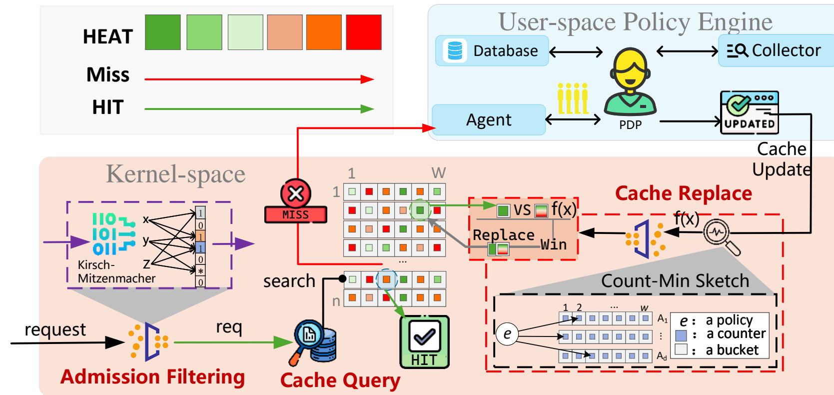


Fig. 1: System Architecture.

Figure 1 illustrates the user–kernel access control architecture for data connectors. The system comprises four core modules: admission filtering, cache query, cache replacement, and user-space policy engine, where the first three modules reside in kernel space.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_27

Admission filtering. When an access request arrives at the kernel enforcement point, a Bloom filter is employed as a sentinel to verify whether the requested resource actually exists in the data connector. If the resource does not exist, the access request is rejected before entering the access control system.

Cache query. After requests pass admission filtering, a hash-indexed lookup is performed in the cache to determine whether a matching rule exists for the request. On a cache hit, the system makes an access control decision based on the matched rule and returns the result directly. On a cache miss, the request is forwarded to the user-space policy engine.

User-space policy engine. Once an access request is forwarded to user space, it is put into a scheduling queue for processing. The policy engine performs complete policy matching and generates the final decision. After producing the verdict, the engine not only returns the grant or deny directive to the connector, but also sends the successfully matched rule to kernel space for caching.

Cache replacement. Upon receiving a new rule from user space, a Count-Min sketch is used to query the approximate access frequency of the new rule, and performs a hash-indexed lookup in the cache to locate the victim entry. The system compares the access frequency of the new rule against that of the victim: the eviction and replacement operation are executed only if the new rule's frequency strictly exceeds that of the victim.

3.4 Admission Filtering

For each incoming request, the system extracts its resource key x and generates k independent indices generated via hash functions. It then probes the Bloom filter at these k positions. If any of the corresponding bits is 0, x is definitively not in the data connector; the request is rejected before entering the access-control pipeline.

To reduce kernel resource consumption, we adopt a hybrid approach, combining WyHash mixing steps with MurmurHash3's `fmix64` as the base hash function, and apply the Kirsch–Mitzenmacher double-hashing technique to derive the k required indices.

Specifically, the index generation logic proceeds as follows: First, compute the 64-bit base hash value $H = \texttt{fmix64}(x)$. Using bitwise operations, split H into two 32-bit hash components $h_1(x)$ and $h_2(x)$, and generate k target indices $g_i(x)$ through linear combination:

$$\begin{aligned} h_1(x) &= H \,\&\, \texttt{0xFFFFFFFF}, \quad h_2(x) = H \gg 32 \\ g_i(x) &= (h_1(x) + i \cdot h_2(x)) \pmod m, \quad i = 0, \dots, k-1 \end{aligned} \tag{1}$$

where m denotes the length of the Bloom filter bit array.

Since Bloom filters do not support deletions, we use an eBPF map-in-map double-buffering design: the access request queries a read-only active filter while the control plane rebuilds a standby snapshot, and an atomic pointer swap switches versions without disrupting queries.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_27

3.5 Cache Replacement

Upon receiving a new rule x from user space, a Count-Min sketch is used to query and obtain the approximate access frequency $\widehat{f}(x)$ in constant time and increment the corresponding counters. To prevent cold rules from entering the cache prematurely, insertion is gated by a minimum frequency threshold: a rule x is eligible for admission only if $\widehat{f}(x)$ exceeds a configured floor value.

To strictly comply with eBPF instruction count and stack space constraints, the kernel cache adopts a set-associative structure. The entire cache space is partitioned into n independent sets, each holding W slots, for a total capacity of $N_c = n \times W$. An incoming rule x is mapped to a target set by computing the index $i = \text{fmix64}(x) \bmod n$. If the target set contains an empty slot, the rule is inserted directly. Otherwise, the system iterates over all W slots to identify the entry v with the minimum estimated frequency $\widehat{f}(v)$ as the eviction candidate. Eviction proceeds only if $\widehat{f}(x) > \widehat{f}(v)$, ensuring that an incoming rule must demonstrate higher observed frequency than the weakest resident before displacing it.

To prevent historically frequent entries from occupying cache slots indefinitely, the system applies periodic frequency decay: after every T accesses, a right shift ($c \leftarrow c \gg 1$) is applied to all counters in the Count-Min sketch, halving their values and allowing stale hot entries to become eviction candidates over time.

4 Queueing Analysis and Parameter Optimization

As described above, the sizes of the Bloom filter and the rule cache impact the efficiency of admission filtering and cache lookup, respectively. Let M_b denote the memory allocated to a single Bloom filter instance and M_c denote the memory allocated to the rule cache. To support non-blocking dynamic updates of the Bloom filter, two instances are maintained via a double-buffering scheme. Given a total kernel memory budget M_{limit}, the allocation must satisfy

$$2M_b + M_c \leq M_{\text{limit}}, \quad M_b > 0,\ M_c > 0. \tag{2}$$

Auxiliary structures (e.g., eBPF map metadata and control-flow parameters) incur negligible memory overhead compared with the data-plane state and are therefore omitted from our model.

To determine parameters M_b and M_c, we formulate the system execution as a two-stage queueing model. In the first stage, requests are either intercepted by the Bloom filter or matched in the rule cache, departing immediately. Consequently, an incoming request is forwarded to user space in the second stage with probability P_{miss}. We assume that access control requests arrive according to a Poisson process [21] with rate λ. Let T_k denote the mean kernel-side processing time, including the overhead for cache misses. In practice, the user-space service time (i.e., the combined cost of policy lookup and evaluation) is approximately constant[1]; we denote it by T_u. The user-space queue is therefore modeled as

[1] `https://v1.casbin.org/docs/en/benchmark`

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_27

an M/D/1 queue with effective arrival rate $\lambda_{\text{eff}} = \lambda P_{\text{miss}}$ and utilization $\rho = \lambda P_{\text{miss}} T_u$. The system reaches steady state only when $\rho < 1$. By the Pollaczek–Khinchine formula, the expected waiting time in user space is

$$E[W_u] = \frac{\lambda_{\text{eff}}\, T_u^2}{2(1-\rho)}.$$

The end-to-end expected latency $E[T]$ per request combines kernel-side processing with the user-space sojourn time, defined as follows:

$$E[T] = T_k + P_{\text{miss}}\,(T_u + E[W_u]) = T_k + P_{\text{miss}}\left(T_u + \frac{\lambda P_{\text{miss}} T_u^2}{2\,(1-\lambda P_{\text{miss}} T_u)}\right). \quad (3)$$

Equation (3) shows that $E[T]$ increases monotonically with P_{miss}, so reducing P_{miss} is the primary lever for improving end-to-end performance.

Under a fixed memory budget, partitioning the space between M_b and M_c introduces a fundamental trade-off that dictates P_{miss}. As M_b increases, the Bloom filter false positive rate $\epsilon(M_b)$ decreases, thereby reducing the likelihood of invalid-traffic penetration. However, a large M_b leaves a smaller budget for M_c, which lowers the cache hit rate $H(M_c)$ and increases the probability that access requests are forwarded to user space. Let γ denote the fraction of invalid and long-tail traffic in the workload. Then P_{miss} decomposes as

$$P_{\text{miss}}(M_b, M_c) = \gamma\,\epsilon(M_b) + (1-\gamma)\big(1 - H(M_c)\big). \quad (4)$$

Given N_b rules inserted into the Bloom filter, a bit array of length $m = 8M_b$, and k hash functions, the false positive rate is

$$\epsilon(M_b) = \left(1 - e^{-kN_b/m}\right)^k.$$

Let M_{entry} denote the memory consumed by a single cached rule. The cache holds at most $N_{\text{c}} = \min\{N, \lfloor M_c/M_{\text{entry}} \rfloor\}$ entries. Assuming the N_{c} cached rules correspond to the most frequently accessed rules, the cache hit rate under a Zipf distribution with exponent α is

$$H(M_c) = \frac{\sum_{i=1}^{N_{\text{c}}} i^{-\alpha}}{\sum_{j=1}^{N} j^{-\alpha}}.$$

The parameter configuration problem under a total memory budget M_{limit} is formulated as

$$\begin{aligned} \underset{M_b,\, M_c}{\text{minimize}} \quad & E[T]\big(P_{\text{miss}}(M_b, M_c)\big) \\ \text{subject to} \quad & 2M_b + M_c \le M_{\text{limit}}, \quad M_b > 0,\ M_c > 0. \end{aligned} \quad (5)$$

Substituting the equality $M_c = M_{\text{limit}} - 2M_b$ reduces this to a one-dimensional problem over M_b. Under the stability condition $\rho = \lambda P_{\text{miss}} T_u < 1$, differentiating

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_27

Equation (3) gives $\frac{dE[T]}{dP_{\text{miss}}} > 0$, so minimizing $E[T]$ is strictly equivalent to minimizing P_{miss}.

To enable analytic differentiation, we replace the floor operation $\lfloor \cdot \rfloor$ with a continuous approximation. Applying the chain rule and setting $\frac{dP_{\text{miss}}}{dM_b} = 0$ yields the balance condition that the optimal allocation must satisfy:

$$\gamma \cdot \left| \frac{\partial \epsilon}{\partial M_b} \right| = 2(1-\gamma) \cdot \left| \frac{\partial H}{\partial M_c} \right| . \tag{6}$$

Under the continuous approximation, $\epsilon(M_b)$ is convex and monotonically decreases in M_b, while the cache miss rate $1 - H(M_c)$ increases monotonically as M_b grows and M_c shrinks.

5 Evaluation

5.1 Experimental Setup

Testbed. All experiments are conducted on a FusionServer equipped with an Intel Xeon 8562Y processor (32 physical cores at 2.80 GHz) and 256 GB DDR4 memory, running Ubuntu 24.04 with Linux kernel 6.8. The file system is configured as a memory-backed tmpfs.

System Parameters. To simulate resource-constrained edge data connector deployments, the kernel-side total memory budget M_{limit} is capped at 16 MB unless otherwise noted. The access control model follows the Basic Model[2]. Each set in the kernel cache holds $W = 4$ entries, and the Bloom filter uses $k = 7$ hash functions.

Workload Generation. A synthetic load generator emulates data exchange traffic with a policy space of $N = 10^6$ rules. Access control requests follow a Zipf distribution. Each experimental run issues $r = 10^7$ requests; all measurements are repeated 50 independent times and averaged to suppress scheduling noise.

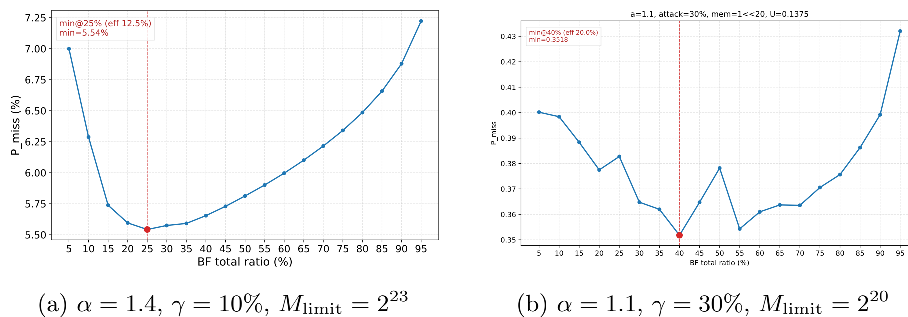
5.2 Experiment 1: Parameter Configuration Validation

Let $\eta = 2M_b/M_{\text{limit}}$ denote the fraction of total memory allocated to the Bloom filter. We conduct a full grid sweep over Zipf skewness $\alpha \in \{1.0, 1.1, \ldots, 2.0\}$, the fraction of tail and invalid traffic $\gamma \in \{10\%, 20\%, \ldots, 90\%\}$, and the total memory budget $M_{\text{limit}} \in \{2^{20}, 2^{21}, \ldots, 2^{24}\}$ bytes, yielding 495 configurations in total. For each configuration, P_{miss} is measured at step size $\Delta\eta = 5\%$ over $\eta \in [5\%, 95\%]$.

The complete collection of 495 sweep curves is available online[3]. Results show that the global minimum of P_{miss} consistently falls within the feasible interior across all configurations, confirming the existence of a non-trivial optimal allocation between the Bloom filter and the cache. Among all configurations,

[2] https://github.com/ukac-paper/ukac/blob/main/config/basic_model.conf
[3] https://github.com/ukac-paper/ukac/tree/main/scripts/image

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_27



(a) $\alpha = 1.4$, $\gamma = 10\%$, $M_{\text{limit}} = 2^{23}$ (b) $\alpha = 1.1$, $\gamma = 30\%$, $M_{\text{limit}} = 2^{20}$

Fig. 2: U-curve behavior of P_{miss} as a function of η under two representative workload and memory configurations.

234 (47.3%) exhibit a strictly unimodal objective, while the remaining 261 (52.7%) contain two or more local minima whose P_{miss} values lie within a small neighborhood of the global optimum. This behavior originates from two discrete effects: the floor operation in cache slot allocation ($N_c = \lfloor M_c/M_{\text{entry}} \rfloor$) and hash-collision granularity in small Bloom filters, both of which induce local non-monotone perturbations near the valley of the objective function.

Figure 2 shows two representative configurations: a well-conditioned strictly unimodal case (α=1.4, γ=10%, M_{limit}=2^{23}) and a case with pronounced local fluctuations (α=1.1, γ=30%, M_{limit}=2^{20}), where four local minima exhibit a maximum relative deviation of 7.30% while the macro-level U-shaped trend remains intact.

5.3 Experiment 2: End-to-End System Performance

We compare against two baselines: Casbin, a widely used user-space access control library, and AppArmor, a kernel-enforced mandatory access control mechanism. Because AppArmor cannot load 10^6 policies natively, we decouple policy-loading overhead from decision latency by organizing the comparison into two aligned groups.

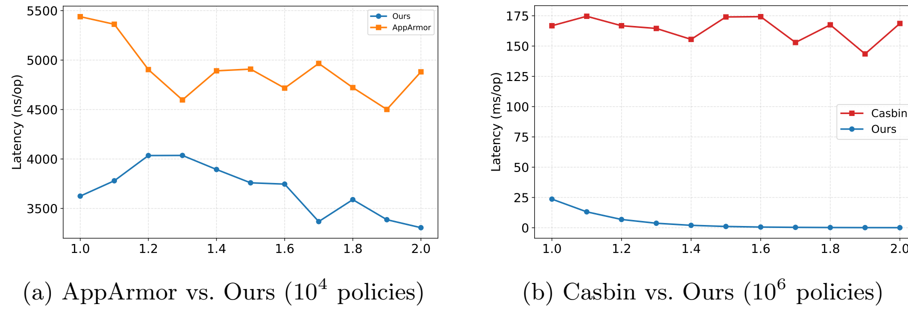
In the first group, both AppArmor and our system use a policy scale of 10^4 rules. Since AppArmor preloads policies into the kernel by design, we pre-populate the corresponding 10^4 rules into the kernel cache of our system to ensure a fair comparison.

In the second group, both Casbin and our system operate at the full scale of 10^6 rules. Since Casbin performs dynamic user-space decisions without preloading, our system starts each run with an empty cache to align the evaluation baseline.

The primary metric is mean service time $E[T]$. By varying the Zipf skewness α, we measure $E[T]$ under different locality conditions; results are presented in Figure 3.

Experimental results show that AppArmor's decision latency falls in the range 4.5–5.4 μs, while our system sustains 3.3–4.0 μs (mean 3.7 μs), a reduction of

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_27



(a) AppArmor vs. Ours (10^4 policies) (b) Casbin vs. Ours (10^6 policies)

Fig. 3: Mean service time $E[T]$ under varying Zipf skewness α for each system.

approximately 25%. Under 10^6 policies with a cold-start configuration, the pure user-space system Casbin yields $E[T] \approx 164.5$ ms and is largely insensitive to α. By contrast, our system's $E[T]$ decays sharply as α increases. At $\alpha = 1.0$, low cache hit rates force a large fraction of requests onto the slow path, yielding $E[T] = 23.65$ ms, which nonetheless represents an 85.6% reduction over Casbin. At $\alpha = 2.0$, hot-spot requests consistently hit the kernel cache, bringing end-to-end latency down to 0.081 ms.

5.4 Experiment 3: Resistance to Tail and Invalid Traffic

We evaluate system behavior across a two-dimensional parameter space defined by Zipf skewness $\alpha \in [1.0, 2.0]$ and the fraction of tail and invalid traffic $\gamma \in [10\%, 100\%]$. Four caching strategies are compared: unprotected LRU, admission filtering with LRU (AF+LRU), cache replacement only (CR), and our full AF+CR scheme. Performance is measured by $E[T]$ and reported as heatmaps in Figure 4, where color intensity encodes latency magnitude.

As shown in Figure 4, standard LRU suffers severe cache thrashing when $\gamma > 40\%$: even under high locality ($\alpha = 2.0$), legitimate hot rules are frequently evicted, causing $E[T]$ to rise substantially. AF+LRU blocks penetration by invalid traffic but admits valid yet infrequent scan requests, which pollute the cache. Conversely, CR mitigates pollution through frequency-aware admission and eviction control but lacks a filter for invalid requests, so $E[T]$ grows under heavy invalid traffic. Our AF+CR scheme maintains the core working set stably in the kernel even under weak locality and intensive attack conditions ($\alpha \approx 1.2$, $\gamma > 80\%$), achieving a reduction in $E[T]$ of up to 65% relative to the LRU baseline.

6 Conclusion

This paper presents a user–kernel access control scheme for lightweight data connectors. A Bloom-filter-based admission filter intercepts invalid traffic before it enters the policy pipeline, and a set-associative frequency-aware cache

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_27

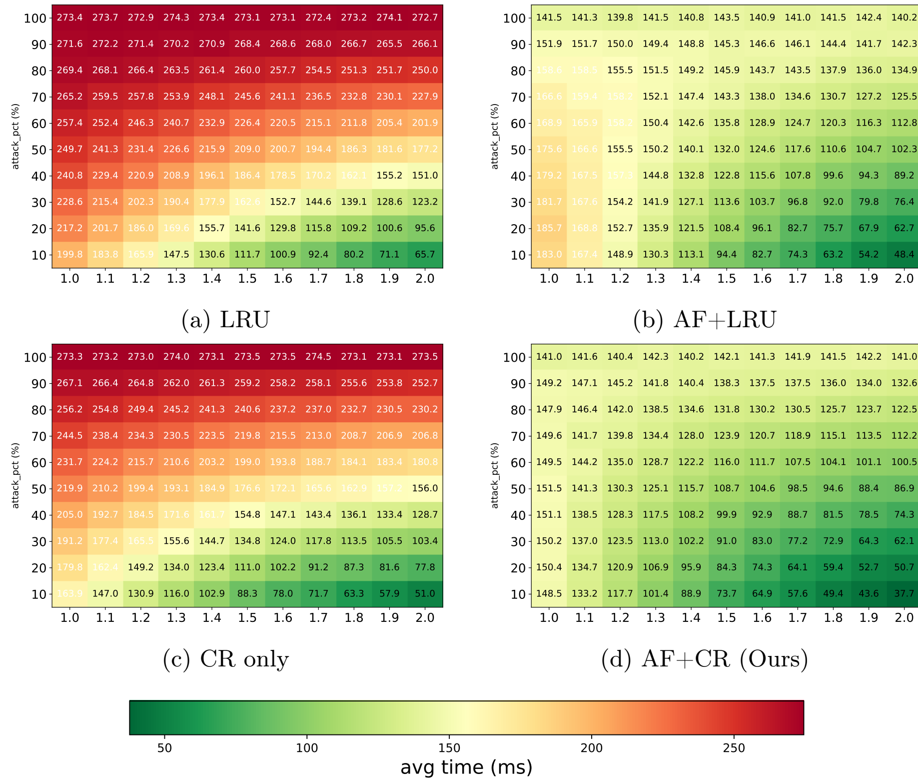


Fig. 4: Heatmaps of $E[T]$ under varying α and attack ratio γ for four caching strategies. The colorbar is shared across all subfigures.

replacement mechanism prevents long-tail requests from polluting the kernel cache. To allocate kernel memory between these two components, we model the user–kernel interaction as an M/D/1 queue and reduce the configuration problem to a one-dimensional search solved by golden-section search. Experimental results show that our scheme achieves lower latency than AppArmor and Casbin across varying workload skewness, and maintains stable performance under intensive invalid and long-tail traffic.

7 Acknowledgements

This work was supported by the National Key Research and Development Program of China (No.2023YFB3106500).

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_27

References

1. Abranches, M., Michel, O., Keller, E., Schmid, S.: Efficient network monitoring applications in the kernel with ebpf and xdp. In: 2021 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN). pp. 28–34 (2021). https://doi.org/10.1109/NFV-SDN53031.2021.9665095
2. Aho, A.V., Denning, P.J., Ullman, J.D.: Principles of optimal page replacement. Journal of the ACM (JACM) **18**(1), 80–93 (1971)
3. Alviano, M., Sestito, P.: User armor: An extension for apparmor. Algorithms **18**(4), 185 (2025)
4. Amitu, D.M., Akol, R.N., Serugunda, J.: Hybrid access control mechanism for massive machine type communications. Discover Internet of Things **5**(1) (2025)
5. Beckmann, N., Chen, H., Cidon, A.: {LHD}: Improving cache hit rate by maximizing hit density. In: 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18). pp. 389–403 (2018)
6. Broder, A., Mitzenmacher, M.: Network applications of bloom filters: A survey. Internet mathematics **1**(4), 485–509 (2004)
7. Cao, L., Meng, L., Stefan, D., Fernandes, E.: Stateful least privilege authorization for the cloud. In: Proc. 33rd USENIX Security Symp. p. 195 (2024)
8. Chen, B., Shen, Q., Xue, L., She, J., Zhang, X., Luo, X., Zhang, X., Chen, W., Wu, Z.: Sack: Enabling environmental situation-aware access control for vehicles in linux kernel. In: 2025 Design, Automation & Test in Europe Conference (DATE). pp. 1–7 (2025)
9. Cheng, Y., Park, J., Sandhu, R.S.: An access control model for online social networks using user-to-user relationships. IEEE Trans. Dependable Secur. Comput. **13**(4), 424–436 (2016)
10. Cormode, G., Muthukrishnan, S.: An improved data stream summary: the count-min sketch and its applications. Journal of Algorithms **55**(1), 58–75 (2005)
11. Cotrini, C., Corinzia, L., Weghorn, T., Basin, D.: The next 700 policy miners: A universal method for building policy miners. In: Proc. ACM SIGSAC Conf. Computer and Communications Security (CCS). pp. 95–112 (2019)
12. Deng, Q., Xu, Z., Zhou, Q., Zhang, Y.: Cordon: Enhancing security through kernel-level control in containerized computing environments. Computers & Security **158**, 104644 (2025)
13. Eiers, W., Sankaran, G., Li, A., O'Mahony, E., Prince, B., Bultan, T.: Quantifying permissiveness of access control policies. In: 44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022. pp. 1805–1817. ACM (2022)
14. Einziger, G., Friedman, R., Manes, B.: Tinylfu: A highly efficient cache admission policy. ACM Transactions on Storage (ToS) **13**(4), 1–31 (2017)
15. Jung, C., Kim, S., Jang, R., Mohaisen, D., Nyang, D.: A scalable and dynamic acl system for in-network defense. In: Proc. 2022 ACM SIGSAC Conf. Computer and Communications Security. pp. 1679–1693 (2022)
16. Kim, J., Park, J., Lee, Y., Song, C., Kim, T., Lee, B.: Petal: Ensuring access control integrity against data-only attacks on linux. In: Proc. 2024 ACM SIGSAC Conf. Computer and Communications Security. pp. 2919–2933 (2024)
17. Li, F., Li, Z., Han, W., Wu, T., Chen, L., Guo, Y.: Cyberspace-oriented access control: Model and policies. In: 2017 IEEE Second International Conference on Data Science in Cyberspace (DSC). pp. 261–266 (2017)

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_27

18. Loscocco, P., Smalley, S.: Integrating flexible support for security policies into the linux operating system. In: Proc. USENIX ATC (2001)
19. Luo, Y., Shen, Q., Wu, Z.: Pml: An interpreter-based access control policy language for web services. arXiv preprint arXiv:1903.09756 (2019)
20. Luo, Y., Shen, Q., Wu, Z.: Perm: Streamlining cloud authorization with flexible and scalable policy enforcement. IEEE Transactions on Information Forensics and Security **20**(7), 8481–8496 (2025)
21. Madkaikar, G., Sural, S., Vaidya, J., Atluri, V.: Queuing theoretic analysis of dynamic attribute-based access control systems. In: Pitropakis, N., Katsikas, S., Furnell, S., Markantonakis, K. (eds.) ICT Systems Security and Privacy Protection. pp. 323–337. Springer Nature Switzerland, Cham (2024)
22. Megiddo, N., Modha, D.S.: {ARC}: A {Self-Tuning}, low overhead replacement cache. In: USENIX Conference on File and Storage Technologies (2003)
23. Shen, B., Shan, T., Zhou, Y.: Improving logging to reduce permission over-granting mistakes. In: Proc. 32nd USENIX Security Symp. pp. 409–426 (2023)
24. Thakare, A., Lee, E., Kumar, A., Nikam, V.B., Kim, Y.G.: Parbac: Priority-attribute-based rbac model for azure iot cloud. IEEE Internet of Things Journal **7**(4), 2890–2900 (2020)
25. Varshith, H.O.S., Sural, S., Vaidya, J., Atluri, V.: Efficiently supporting attribute-based access control in linux. IEEE Trans. Dependable and Secure Comput. **21**(4), 2012–2026 (2024)
26. Wang, H., Fang, Y., Liu, Y., Jin, Z., Delph, E., Du, X., Liu, Q., Xing, L.: Hidden and lost control: on security design risks in iot user-facing matter controller. In: 32nd Annual Network and Distributed System Security Symposium, NDSS 2025, San Diego, California, USA, February 24-28, 2025. The Internet Society (2025)
27. Wazan, A.S., Chadwick, D.W., Venant, R., Billoir, E., Laborde, R., Ahmad, L., Kaiiali, M.: Rootasrole: a security module to manage the administrative privileges for linux. Computers & Security p. 102983 (2022)
28. Xu, S., Wang, Y., Lei, L., Sun, K., Jing, J., Ma, S., Wang, J., Huang, H.: Condo: Enhancing container isolation through kernel permission data protection. IEEE Transactions on Information Forensics and Security **19**, 6168–6183 (2024)
29. Yang, J., Zhang, Y., Qiu, Z., Yue, Y., Vinayak, R.: Fifo queues are all you need for cache eviction. In: Proceedings of the 29th Symposium on Operating Systems Principles. pp. 130–149 (2023)
30. Zhou, Y., Philbin, J., Li, K.: The multi-queue replacement algorithm for second level buffer caches. In: USENIX Annual Technical Conference, General Track. pp. 91–104 (2001)
31. Zhu, S., Zhang, Y.: Probabilistic access policies with automated reasoning support. In: Proc. Int. Conf. Computer Aided Verification (CAV). pp. 443–466 (2024)

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_27