computers

MDPI

Article

Least-Privilege Role-Based Access Control Improvement for Cloud Container Security

Waleed K. Abdulraheem [1,*], Emad Mohammed Ibbini [2], Hasan Kanaker [3], Sami Smadi [4], Nader Abdel Karim [1], Hussam N. Fakhouri [1], Layla Albdour [5] and Sandi Fakhouri [6]

[1] Department of Intelligent Systems, Faculty of Artificial Intelligence, Al-Balqa Applied University, Al-Salt 19117, Jordan; nader.salameh@bau.edu.jo (N.A.K.); h.fakhouri@bau.edu.jo (H.N.F.)
[2] Department of Computer Science, Faculty of Information Technology, Al al-Bayt University, Mafraq 25113, Jordan; ibbini4@aabu.edu.jo
[3] Department of Information Security, Faculty of Information Technology, University of Petra, Amman 11942, Jordan; hasan.kanaker@uop.edu.jo
[4] Department of Information Technology, Faculty of Information Technology and Computer Sciences, Yarmouk University, Irbid 21163, Jordan; sami.smadi@yu.edu.jo
[5] Cyber Security Department, Faculty of Science and Information Technology, Al-Zaytoonah University of Jordan, Amman 11733, Jordan; l.albdour@zuj.edu.jo
[6] Department of Computer Science, School of Information Technology, The University of Jordan, Amman 11942, Jordan; sandi_fakhouri@uoj.edu.jo
* Correspondence: w.khalid@bau.edu.jo

Abstract

Role-Based Access Control (RBAC) is the de-facto mechanism for preserving Kubernetes and other cloud-native container platforms, however real deployments occasionally drift away from the principle of least privilege as clusters, teams, and services improve. This paper introduces an automated RBAC hardening framework that formulates least-privilege policy design as a limited optimization problem over RoleBindings and ClusterRoleBindings. The objective combines (i) a permission-risk score for namespaced and cluster-scoped actions with (ii) an operational complexity term that discourages overly large binding sets. Solid limitations encode functional requirements as well as practical security policies, which includes namespace allowlists, role scoping rules, administrative restrictions on cluster-wide bindings, binding budgets, and separation-of-duty requirements expressed by utilizing capability classes. To allow optimizer-agnostic search while protecting Kubernetes RBAC semantics, we analyze candidate policies by utilizing a unified penalty-based fitness function that compines risk, complexity, and constraint violations into a single scalar value. We utilized ten metaheuristic as a benchmark including baseline search paths on a Kubernetes-inspired instance and report feasibility and least-privilege quality metrics (precision, recall, F1, and over-privilege ratio) parallel to RB/CRB counts and excess risk as a structural indicators. Outcomes present that feasibility is the prime challenge, and is restricted to a subset of optimizers reliably arrives to entirely feasible and compact arrangements within the exact budget, indicating the practicality of metaheuristic enhancement for systematic RBAC reduction in containerized cloud computing environments.

Keywords: least-privilege; Role-Based Access Control (RBAC); kubernetes; cloud computing container; container security; metaheuristic optimization; policy hardening; constraint satisfaction

Check for updates

Academic Editors: Leandros Maglaras and Helge Janicke

Received: 25 March 2026
Revised: 5 May 2026
Accepted: 12 May 2026
Published: 21 May 2026

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

1. Introduction

Principles of least privilege (PoLP) and Role-Based Access Control (RBAC) are well-instituted foundations for minimizing redundant authority in software systems: users should only be granted the authorizations needed for their tasks, and permissions are normally administered through roles rather than through ad-hoc per-user grants [1–3]. In Kubernetes and other cloud-native environments, however, due to the frequent changes within service accounts, namespaces, API resources, and operational requirements this principle is difficult to maintain; administrators often respond by granting broad roles, which protects availability but escalates the impact of compromise or unlawful access [4,5].

Prior research has discussed vital components of this problem, including RBAC role mining and role minimization [6–8], least-privilege policy enhancement for cloud access control [4,9], and container-security techniques which include image hardening, runtime monitoring, and isolation enhancement [5,10]. The remaining gap is that these branches occasionally separately treat role discovery, policy recommendation, and container hardening. They do not offer a unified, optimizer-agnostic formulation that reconfigures Kubernetes RoleBindings and ClusterRoleBindings directly while maintaining needed access and enforcing implementing limitations such as namespace allowlists, binding budgets, administrative approval for cluster-wide bindings, and separation-of-duty principles.

This paper handles this gap by formulating Kubernetes-oriented RBAC hardening as a limited optimization problem. The novelty is the integration of Kubernetes RBAC semantics, explicit operational constraints, a risk–complexity objective, and a unified penalty-based analyzer that permits various black-box metaheuristic algorithms to scope the same policy space without altering the foundational access-control semantics. The proposed formulation is directed as a systematic least-privilege hardening aspect: it can be operated on a current RBAC snapshot and, as discussed later, can be integrated in a ongoing adaptive loop for dynamic clusters.

This paper introduces the following contributions:

- A risk-aware least-privilege RBAC formulation for cloud containers that collaboratively minimizes permission risk and binding complexity while Adequate functional needs.
- A direct encoding of practical Kubernetes-style limitations, containing namespace allowlists, role-scoping rules, administrative limitations on ClusterRoleBindings, binding budgets, and separation-of-duty policies across capability classes.
- An integrated penalty-based fitness function that rebuilds effective privileges from candidate bindings and transforms all constraint violations into a single scalar value, enabling adequate benchmarking of heterogeneous metaheuristics.
- An empirical evaluation of ten optimizers under optimal budgets, demonstrating that feasibility is the primary challenge and that only a subset of optimizers consistently finds compact, low-risk RBAC settings.

Paper Organization

The rest of the paper is structured as listed below: Section 2 assesses RBAC fundementals, least-privilege role engineering, and cloud computing containers and its security. Section 3 presents the least-privilege RBAC optimization problem, variables, constraints, and the penalized fitness designs. Section 4 presents the optimization framework and the frequent assessment pipeline used to benchmark various metaheuristics. Section 5 illustrates the benchmark instance, optimizers, experimental protocol, and examination metrics. Section 6 demonstrates quantitative results and evaluates feasible and infeasible activities among optimizers. Section 7 lastly analyzes main findings, limitations, and practical implications for implementable RBAC hardening.

https://doi.org/10.3390/computers15050326

2. Related Work

The conceptual foundation for this work is provided by the RBAC research. Standard RBAC models separate subjects, roles, permissions, and constraints, thereby enhancing protocol manageability in comparison with direct per-subject permission assignment [2,3]. Role engineering or role mining is a major pertinent line, where roles are implied from existing user–permission assignments. This issue is computationally complicated, and previous studies have proposed exact, heuristic, Boolean-matrix, semantic, and multi-objective paths to achieve compact role sets [6–8,11–14].

Role mining is extended through Least-privilege policy refinement by emphasizing the removal of irrelevant grants while maintaining necessary functionality. Current studies acknowledges privilege-assignment errors, perturbation-aware migration, attribute-based least-privilege mining, user-role recommendation, and automatic privilege reduction for access-control policies [4,9,15–19]. These studies promote optimization-based treatment of least privilege, however, they occasionally do not model Kubernetes-specific binding structures together with operational limitations such as namespace scope, cluster-wide binding approval, binding budgets, and separation-of-duty classes.

Complementary risks, which include insecure images, software supply-chain integrity, runtime monitoring, container escape detection, isolation, and adaptive defense are addressed through container and cloud-native security research [5,10,20–27]. Studies within the areas of Kubernetes migration and operational-security studies also present that availability, scalability, and security must be handled together in production orchestration environments [28–30]. These studies minimize container risk at other layers, while the current paper centers its focus specifically on RBAC policy minimization.

Contrary to prior studies, the presenteed framework treats least-privilege Kubernetes RBAC design as a constrained search issue over RoleBindings and ClusterRoleBindings. It maintains required permissions, penalizes incorrect or unsafe configurations, and evaluates numerous metaheuristic optimizers by utilizing the same decoder and fitness function. This positioning illustrates the research gap addressed by the paper: automated, constraint-aware RBAC hardening for cloud container environments in comparison to general role discovery or standalone container hardening.

Table 1 clarifies the practical distinction between the proposed approach and representative access-control alternatives. Advantageous for administration and role discovery are the approaches of Classical RBAC, however they do not directly optimize Kubernetes RoleBindings/ClusterRoleBindings under namespace, cluster-scope, binding-budget, and separation-of-duty constraints.The proposed formulation therefore has a functional intendment: it establishes a concrete candidate policy that can be audited and launched after validation, at the same time quantifying how much over-privilege and excess risk were eliminated.

Table 1. Comparison between the proposed Kubernetes-oriented RBAC optimization framework and representative RBAC/access-control approaches.

Approach	Main Purpose	Difference from This Work
Standard RBAC administration	Manually assign users or service accounts to predefined roles	Does not explore for a lower-risk binding set and can protect over-broad permissions.
Role mining/minimization	Infer compact roles from existing user–permission matrices	Optimizes role structure, but typically abstracts away Kubernetes RoleBindings, ClusterRoleBindings, namespace scope, and operational security limitations.
Least-privilege refinement/recommendation	Recommend or remove excessive permissions from access-control policies	Enables least privilege, but usually does not support an optimizer-independent constrained representation for Kubernetes-style RBAC objects.

Table 1. *Cont.*

Approach	Main Purpose	Difference from This Work
Container hardening/ runtime security	Reduce container, image, runtime, or orchestration attack surfaces	Targets complementary layers; it does not explicity redesign RBAC bindings or quantify least-privilege policy standard.
Proposed framework	Optimize Kubernetes-style RBAC bindings under functional and security constraints	Specifically searches RoleBindings/ClusterRoleBindings, reports feasibility and least-privilege metrics, and yields an auditable optimized policy.

3. Least-Privilege RBAC Optimization for Cloud Container Security Problem Description

Access control is imposed through Roles, ClusterRoles, RoleBindings, and ClusterRoleBindings within a Kubernetes-like cloud container environment. Within a specific namespace, individual role collects permissions (verbs on API resources), while ClusterRoles can allow permissions within the entire cluster. In order to determine what tasks each subject is permitted to execute, bindings attach these roles to subjects, such as human users, groups, or service accounts.

The formulation beneath is correlated to a fixed RBAC snapshot expressed by the current subjects, roles, permissions, workload requirements, and security policies. Under identical inputs this snapshot abstraction is effective for determinig the constrained optimization issue and for benchmarking optimizers. In an operational deployment the same model can be re-instantiated whenever manifests, audit-log evidence, service accounts, namespaces, custom resources, or API permissions are altered. It should not be interpreted as assuming that Kubernetes clusters are fixed.

The security goal in this environment is to enforce an RBAC configuration that concurrently enables all functional requirements and meets strict security restrictions. The necessary actions must be executed through every subject in the appropriate namespaces or at the cluster domain, while the implementation must abide to strict security policies which include namespace allowlists, role scoping rules, administrative restrictions on ClusterRoleBindings, binding budget limits, and separation-of-duty restrictions. Simultaneously, the configuration should reduce overall security risk by reducing unnecessary or excessive privileges, complying as closely as possible to the least-privilege principle, and continue to be operationally simple by avoiding an unnecessary number of bindings. The flow of the proposed least-privilege RBAC optimization loop is shown in Figure 1.

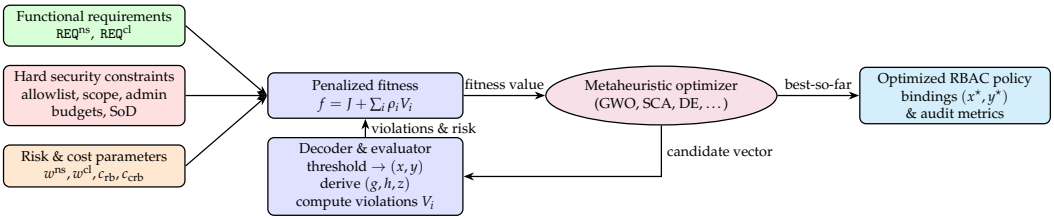


Figure 1. Overview of the proposed least-privilege RBAC optimization loop: requirements, constraints, and risk/complexity elements feed a penalized fitness; a metaheuristic proposes candidate bindings; the assessors reconstructs effective permissions and violations; and the best feasible policy is exported with audit metrics.

We introduce this design task as a constrained single-objective optimization issue. Dealing with the model strictly as a mixed-integer program can be computationally expensive due to the decisions being binary and the constraints being numerous and complicated. Rather, we utilize population-based metaheuristic optimizers (such as GWO, PSO, SCA, and relevant algorithms) along with a unified penalty-based fitness function that encodes

both risk and constraint abuses. Various optimizers to be benchmarked objectively on the same RBAC instance are permitted as a result of this formulation.

3.1. Mathematical Model

3.1.1. Base Objective (Risk and Complexity)

The base objective aggregates risk and configuration complexity:

$$J(x,y,g,h) = \underbrace{\sum_{s\in S}\sum_{n\in N}\sum_{p\in P^{\mathrm{ns}}} w^{\mathrm{ns}}_{s,n,p}\, g_{s,n,p}}_{\text{namespaced risk}} + \underbrace{\sum_{s\in S}\sum_{q\in P^{\mathrm{cl}}} w^{\mathrm{cl}}_{s,q}\, h_{s,q}}_{\text{cluster risk}} + \underbrace{c_{\mathrm{rb}}\sum_{s,n,r} x_{s,n,r} + c_{\mathrm{crb}}\sum_{s,c} y_{s,c}}_{\text{binding complexity}}. \tag{1}$$

Equation (1) is reached through decomposing the cost of an RBAC formulation into the two quantities that the framework is programmed to reduce subsequent functional access is maintained. The primary two summations add the risk of every effective granted permission: namespaced grants are weighted by $w^{\mathrm{ns}}_{s,n,p}$ and cluster-scoped grants are weighted by $w^{\mathrm{cl}}_{s,q}$. The last two summations add a minor operational-complexity cost for each RoleBinding and ClusterRoleBinding so that, the optimizer prefers the simpler policy that is easier to evaluate, among policies with similar risk. Therefore, the base objective is not a generic accuracy metric; it is an explicit least-privilege design cost which consists of permission exposure in addition to policy complexity.

In order to maintain this interpretation, coefficients are selected. In the benchmark, cluster-scoped and sensitive-resource permissions acquire bigger weights than typical namespaced read permissions; RoleBinding and ClusterRoleBinding costs are maintained lower than permission-risk weights so that the optimizer at no time eliminates required security-relevant grants solely to minimize the quantity of bindings. Rather than being universal constants, these weights are configurable inputs.They cause the objective to be transparent, but they also establish subjectivity: varying organizations could possibly rate the same permission differently based on compliance obligations, namespace criticality, incident history, and workload exposure. Therefore, the numerical outcomes should be understood relative to the chosen risk model, and practical deployment should constitute calibration and sensitivity analysis, as discussed in Section 6.7.

3.1.2. Constraints

Permission Derivation (RBAC Semantics)

Namespaced permissions come from RoleBindings in that namespace and from ClusterRoleBindings that refer to clusterRoles with namespaced rules:

$$g_{s,n,p} \le \sum_{r\in R} A^{\mathrm{ns}}_{r,p} x_{s,n,r} + \sum_{c\in C} A^{\mathrm{ns}}_{c,p} y_{s,c}, \quad \forall s,n,p. \tag{2}$$

Additionally, if a binding grants p, then $g_{s,n,p} = 1$, which can be enforced with lower-bound implications; for example,

$$g_{s,n,p} \ge x_{s,n,r} \quad \forall s,n,p,r \text{ such that } A^{\mathrm{ns}}_{r,p} = 1, \tag{3}$$

$$g_{s,n,p} \ge y_{s,c} \quad \forall s,n,p,c \text{ such that } A^{\mathrm{ns}}_{c,p} = 1. \tag{4}$$

Cluster permissions result only from ClusterRoleBindings:

$$h_{s,q} \leq \sum_{c \in C} A^{\text{cl}}_{c,q} y_{s,c}, \quad \forall s, q, \tag{5}$$

$$h_{s,q} \geq y_{s,c}, \quad \forall s, q, c \text{ with } A^{\text{cl}}_{c,q} = 1. \tag{6}$$

Requirement Satisfaction

All required permissions must be granted:

$$g_{s,n,p} \geq \text{REQ}^{\text{ns}}_{s,n,p}, \quad \forall s, n, p, \tag{7}$$

$$h_{s,q} \geq \text{REQ}^{\text{cl}}_{s,q}, \quad \forall s, q. \tag{8}$$

Feasibility Constraints: Allowlist, Scope, Admin

Namespace allowlist:

$$x_{s,n,r} \leq \text{ALLOWNS}_{s,n}, \quad \forall s, n, r. \tag{9}$$

Role scoping (Roles only in their own namespace; ClusterRoles arbitrary):

$$x_{s,n,r} \leq \text{allowedX}_{n,r}, \quad \forall s, n, r. \tag{10}$$

ClusterRoleBindings limited to privileged subjects:

$$\sum_{c \in C} y_{s,c} \leq |C| \, \text{AdminOK}_s, \quad \forall s. \tag{11}$$

Binding Budget Limits

Maximum RoleBindings per subject-namespace pair:

$$\sum_{r \in R} x_{s,n,r} \leq B^{\text{rb}}_{s,n}, \quad \forall s, n. \tag{12}$$

Maximum ClusterRoleBindings per subject:

$$\sum_{c \in C} y_{s,c} \leq B^{\text{crb}}_{s}, \quad \forall s. \tag{13}$$

Separation-of-Duty (SoD)

First relate capability indicators to effective permissions. A subject has capability k if it has any permission in that capability:

$$z_{s,k} \geq \sum_{n \in N} \sum_{p \in P^{\text{ns}}} g_{s,n,p} \, M^{\text{ns}}_{k,p}, \quad \forall s, k, \tag{14}$$

$$z_{s,k} \geq \sum_{q \in P^{\text{cl}}} h_{s,q} \, M^{\text{cl}}_{k,q}, \quad \forall s, k, \tag{15}$$

$$z_{s,k} \leq 1, \quad \forall s, k. \tag{16}$$

For each forbidden pair $(k_1, k_2) \in \mathcal{D}$, we enforce:

$$z_{s,k_1} + z_{s,k_2} \leq 1, \quad \forall s, \ (k_1, k_2) \in \mathcal{D}. \tag{17}$$

3.1.3. Penalized Fitness for Metaheuristics

Instead of solving the mixed-integer model right away, we permit metaheuristics to function on a continuous decision vector and alter it to binary (x, y) via thresholding. Constraint violations are then penalized in the fitness function:

$$f(x, y, g, h, z) = J(x, y, g, h) + \rho_{\text{req}} V_{\text{req}} + \rho_{\text{inv}} V_{\text{inv}} + \rho_{\text{lim}} V_{\text{lim}} + \rho_{\text{sod}} V_{\text{sod}}, \quad (18)$$

such that $V_{\text{req}}, V_{\text{inv}}, V_{\text{lim}}, V_{\text{sod}}$ are non-negative violation measures (numbers of missing permissions, invalid bindings, budget attacks, and SoD attacks consequently). Feasible solutions have zero penalty, so their fitness equals the base objective $J(\cdot)$.

Equation (18) was acquired by converting the constrained optimization model into an exterior-penalty issue. The violation terms enforce hard requirements while the term $J(\cdot)$ ranks feasible policies by risk and complexity. The penalty coefficients are chosen through utilizing a lexicographic standard: a single missing needed permission or invalid binding needs to be more expensive than any plausible enhancement within the base objective. This certifies that the search first satisfies functional and security constraints and only then reduces risk and policy size. In implementation, this can be reached by setting $\rho_{\text{req}}, \rho_{\text{inv}}, \rho_{\text{lim}}, \rho_{\text{sod}}$ one or more orders of magnitude larger than the maximum possible variation in $J(\cdot)$,with the requirement and invalid-binding penalties treated as the highest-priority violations.

3.1.4. Notation

Sets

- S: set of subjects (users/service accounts), indexed by s.
- N: set of namespaces, indexed by n.
- R: set of all roles (Roles and ClusterRoles), indexed by r.
- $C \subseteq R$: subset of clusterRoles, indexed by c.
- P^{ns}: set of namespaced permission atoms, indexed by p.
- P^{cl}: set of cluster-scoped permission atoms, indexed by q.
- K: set of capability classes (e.g., *SECRETS_READ*, *RBAC_WRITE*), indexed by k.

Parameters

- $A^{\text{ns}}_{r,p} \in \{0, 1\}$: 1 if role r includes namespaced permission $p \in P^{\text{ns}}$.
- $A^{\text{cl}}_{c,q} \in \{0, 1\}$: 1 if clusterRole $c \in C$ includes cluster permission $q \in P^{\text{cl}}$.
- $\text{REQ}^{\text{ns}}_{s,n,p} \in \{0, 1\}$: 1 if subject s requires permission p in namespace n.
- $\text{REQ}^{\text{cl}}_{s,q} \in \{0, 1\}$: 1 if subject s requires cluster permission q.
- $\text{ALLOWNS}_{s,n} \in \{0, 1\}$: 1 if subject s is allowed to operate in namespace n.
- $\text{AdminOK}_s \in \{0, 1\}$: 1 if subject s is allowed to receive ClusterRoleBindings.
- $\text{allowedX}_{n,r} \in \{0, 1\}$: 1 if role r may be bound in namespace n (role scoping).
- $B^{\text{rb}}_{s,n}$: maximum number of RoleBindings allowed for subject s in namespace n.
- B^{crb}_{s}: maximum number of ClusterRoleBindings allowed for subject s.
- $M^{\text{ns}}_{k,p} \in \{0, 1\}$: 1 if namespaced permission p belongs to capability class k.
- $M^{\text{cl}}_{k,q} \in \{0, 1\}$: 1 if cluster permission q belongs to capability class k.
- $\mathcal{D} \subseteq K \times K$: set of forbidden capability pairs (k_1, k_2); no subject may hold both.
- $w^{\text{ns}}_{s,n,p} \geq 0$: risk weight for permission p granted to subject s in namespace n.
- $w^{\text{cl}}_{s,q} \geq 0$: risk weight for cluster permission q granted to subject s.
- $c_{\text{rb}} \geq 0$: cost per RoleBinding.
- $c_{\text{crb}} \geq 0$: cost per ClusterRoleBinding.

Penalty Weights

- ρ_{req}: penalty for missing required permissions.
- ρ_{inv}: penalty for invalid bindings (allowlist, scope, admin).
- ρ_{lim}: penalty for binding-budget violations.
- ρ_{sod}: penalty for separation-of-duty violations.

3.1.5. Decision Variables

Binding Decisions

$x_{s,n,r} \in \{0,1\}$ RoleBinding: subject s bound to role r in namespace n,

$y_{s,c} \in \{0,1\}$ ClusterRoleBinding: subject s bound to clusterRole c.

Derived Effective Permissions

$g_{s,n,p} \in \{0,1\}$ 1 if s effectively has permission $p \in P^{\text{ns}}$ in namespace n,

$h_{s,q} \in \{0,1\}$ 1 if s effectively has cluster permission $q \in P^{\text{cl}}$.

Capability Indicators

$z_{s,k} \in \{0,1\}$ 1 if subject s has at least one permission in capability k.

3.2. Flowchart of the Metaheuristic RBAC Optimization

Figure 2 shows the high-level flowchart of the framework: data preprocessing, optimization loop, and result export.

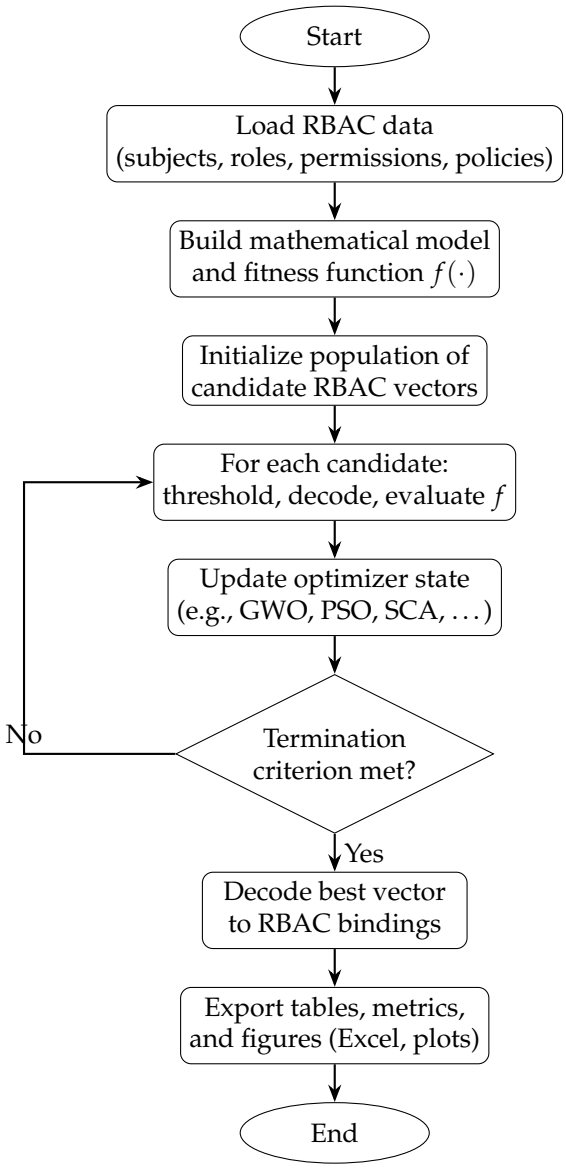


Figure 2. High-level flowchart of the least-privilege RBAC optimization framework.

3.3. Pseudocode and Description

Algorithm 1 conveys one metaheuristic optimizer run (e.g., GWO) on the RBAC issue. Various optimizers share the same interface: population size, iteration budget, bounds, and a handle to the fitness function $f(\cdot)$.

Algorithm 1 Metaheuristic RBAC Optimizer (Generic Skeleton)

Require: Population size N_{pop}, max iterations $T_{\max}$, bounds $\boldsymbol{\ell}, \boldsymbol{u}$, fitness $f(\cdot)$.
Ensure: Best decision vector $\boldsymbol{z}^{\star}$ and its fitness $f^{\star}$.

1: **Input:** RBAC data (subjects, namespaces, roles, permissions, policies)
2: Build model parameters and internal data structures
3: Define fitness function $f(\boldsymbol{z})$ via penalized evaluation ▷ Equation (18)
4: Initialize population $\{\boldsymbol{z}^{(i)}\}_{i=1}^{N_{\text{pop}}}$ uniformly in $[\boldsymbol{\ell}, \boldsymbol{u}]$
5: Initialize optimizer-specific state (e.g., alpha/beta/delta wolves for GWO)
6: **for** $t = 1$ to $T_{\max}$ **do**
7: **for** $i = 1$ to N_{pop} **do**
8: Enforce bounds: $\boldsymbol{z}^{(i)} \leftarrow \max(\boldsymbol{\ell}, \min(\boldsymbol{u}, \boldsymbol{z}^{(i)}))$
9: Threshold to binary: $\boldsymbol{b}^{(i)} \leftarrow \mathsf{Threshold}(\boldsymbol{z}^{(i)})$
10: Decode $(x^{(i)}, y^{(i)})$ from $\boldsymbol{b}^{(i)}$
11: Compute $(g^{(i)}, h^{(i)}, z^{(i)})$ and violations $V_{\text{req}}^{(i)}, V_{\text{inv}}^{(i)}, V_{\text{lim}}^{(i)}, V_{\text{sod}}^{(i)}$
12: Compute fitness $f^{(i)} \leftarrow f(x^{(i)}, y^{(i)}, g^{(i)}, h^{(i)}, z^{(i)})$
13: Update best-known solution $(\boldsymbol{z}^{\star}, f^{\star})$ if $f^{(i)} < f^{\star}$
14: **end for**
15: Update population $\{\boldsymbol{z}^{(i)}\}$ according to metaheuristic rules
16: Optionally, record convergence curve: $F_t \leftarrow f^{\star}$
17: **end for**
18: **return** $(\boldsymbol{z}^{\star}, f^{\star}, \{F_t\})$

3.3.1. Problem Setup

The RBAC dataset is loaded and preprocessed into the internal model factors within the primary stage of the algorithm. This consists of the structure of the namespaced and cluster permission matrices A^{ns} and A^{cl}, the requirement indicators REQ^{ns} and REQ^{cl}, and the different policy aims such as ALLOWNS, AdminOK, binding budgets, and separation-of-duty capability maps. Risk weights w^{ns} and w^{cl} are also started at this stage. Based on these structures, a penalized fitness function $f(\boldsymbol{z})$ is defined, encapsulating both the RBAC semantics and all constraint terms in a single scalar objective.

3.3.2. Initialization

Initialization begins through constructing a population of continuous candidate vectors $\boldsymbol{z}^{(i)}$ within the bounded domain, typically using a uniform distribution over $[\boldsymbol{\ell}, \boldsymbol{u}]$. Each specifc vector represents a flattened version of all binary decisions $x_{s,n,r}$ and $y_{s,c}$ that determine an RBAC formulation. Simultaneously, the optimizer may initialize its own internal state.

3.3.3. Main Loop

The main iterative loop shown in lines 6–18 is the core of the algorithm). At each alteration, each candidate vector is first clipped to the permitted bounds to authenticate feasibility in the continuous search space. To construct a concrete binding configuration, a continuous vector is thresholded to a binary representation. Subsequently, this binary vector is decoded into the role-binding variables x and cluster-role-binding variables y, which are subsequently used to calculate the effective permissions g and h and the capability indicators z.

The evaluator computes the fitness value by gathering the base objective (risk and binding complexity) with all constraint violations, weighted by their penalty factors when the derived quantities are available. The outcome is a single scalar f that returns both security quality and constraint satisfaction. The optimum solution and its related metrics are updated appropriately once the fitness value is compared to the best-so-far solution, and if it is strictly better.

3.3.4. Population Update

Correlating with line 17, the optimizer executes a population update step after all individuals within the population have been calculated and the optimum solution has been updated. The accurate update mechanism depend on the selected metaheuristic: in GWO it is dependent on the relative positions of alpha, beta, and delta wolves; in PSO it integrates personal and global best positions; in SCA it uses sinusoidal transformations, and so on. Importantly, the framework handles this update as a black-box operation that only requires access to the present population and their corresponding fitness values.

3.3.5. Termination and Output

The algorithm repeats to iterate until the largest quantity of iterations $T_{\max}$ is derived, or a different termination position is satisfied (for example, convergence of the fitness values). Ultimately, the optimum decision vector $z^{\star}$ and its fitness $f^{\star}$ are returned, collected along with the convergence history. This best vector is decoded into the last RBAC formulation, as well as the explicit RoleBindings and ClusterRoleBindings, which can then be inspected, exported, and, if suitable, deployed in the target environment.

3.4. Framework Description

The data and model layer is tasked with transforming raw RBAC information into a consistent numerical representation. It receives as input the lists of subjects, namespaces, roles and clusterRoles, permission atoms, and all related security policies. From these inputs, it formulates the permission matrices A^{ns} and A^{cl}, the requirement indicators $\mathrm{REQ}^{\mathrm{ns}}$ and $\mathrm{REQ}^{\mathrm{cl}}$, the namespace allowlists ALLOWNS, the administrative restriction map AdminOK, binding budgets B^{rb} and B^{crb}, capability maps M^{ns} and M^{cl}, and the set of forbidden capability pairs $\mathcal{D}$. This layer creates the real-world RBAC knowledge and limitations, even remains agnostic to the specific optimization algorithm that will utilize these parameters.

The evaluation layer applies the prime mapping from a potential decision vector $f(\boldsymbol{z})$. It primarily implements thresholding to alter the continuous vector into a binary representation, and then decodes that representation into the binding variable x and y. Using these variables, the evaluator recreates the effective permissions g and h, and calculates capability indicators z. It then calculates all relevant violations, including missing required permissions, invalid bindings, binding-budget violations, and separation-of-duty violations. Lastly, it collects these counts with the base risk and binding complexity to create the penalized fitness value. In conjunction to the scalar fitness, the evaluation layer can provide comprehensive diagnostics such as binding counts, violation statistics, risk decomposition, and accuracy metrics such as precision, recall, F1 score, and over-privilege.

One or more metaheuristic algorithms are hosted through the optimizer layer. (such as GWO, PSO, SCA, MFO, HHO, MVO, ALO, BOA, or SSA) that operate on the same RBAC issue. Each optimizer executes its own internal representation of the population and its own update principles, however all optimizers operate with the evaluation layer through a typical interface: they propose candidate vectors, receive fitness values in return, and alter their populations correspondingly. These optimizers are interchangeable black boxes,

within the perspective of the RBAC issue, which permits for reasonable and systematic benchmarking of various algorithms on exactly the same model and data.

The result analysis and export layer processes the outputs created by the optimizers. It decodes the optimum solutions established by each algorithm into explicit RoleBindings and ClusterRoleBindings, and computes aggregate statistics within runs, including the fraction of feasible runs, the mean and standard deviation of the fitness, and the mean values of security metrics like F1 score. This layer also yields tables and figures to external formats, for example creating Excel workbooks and graphical plots of convergence curves, boxplots, and bar charts. These externals form the basis for scientific reporting, which include LaTeX tables and figures summarizing performance and security characteristics of the optimized RBAC configurations.

Figure 3 illustrates the conceptual framework and the interactions between the layers. The data and model layer provides the evaluation layer with the essential parameters. The evaluation layer provides fitness values and receiving candidate solutions by interacting bidirectionally with the optimizer. Ultimately, the evaluation outputs are transferred on to the result analysis and export layer.

Instead of being executed as a one-time tool, the same layers can be executed as recurring hardening loops for production use. Periodically, the data layer refreshes inputs from manifests, the live RBAC state, audit logs, and policy-as-code rules; the optimizer nominates a candidate RBAC update; the evaluation layer inspects functional coverage and security limitations; and the outcome layer exports a minimal change set for dry-run validation, canary deployment, rollback monitoring, and operator approval. The Conceptual view of the search landscape is shown in Figure 4. An Interpretation of the compact feasible GWO/SCA policy in the benchmark instance is shown in Table 2.

Table 2. Interpretation of the compact feasible GWO/SCA policy in the benchmark instance. The RB = 3 and CRB = 1 result is sufficient only for this three-subject, two-namespace benchmark and should not be generalized as a universal number of Kubernetes roles.

Subject	Scope	Optimized Binding	Practical Purpose
ci-bot	*dev* namespace	RoleBinding to a development deployer role	Allows CI/CD actions required for development deployment without granting production or cluster administration authority.
frontend-sa	*prod* namespace	RoleBinding to a frontend runtime role	Allows the frontend workload to access only the production resources needed at runtime.
auditor	*prod* namespace	RoleBinding to a namespace read/audit role	Permits inspection of production resources without write privileges.
auditor	Cluster scope	ClusterRoleBinding to a read-only cluster inspection role	Permits cluster-level read-only inspection needed by the benchmark while escaping privileged administrative capabilities.

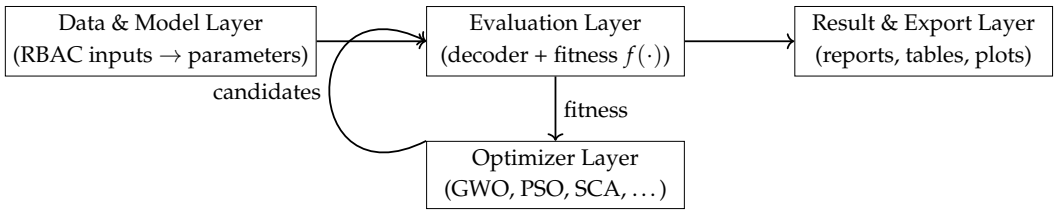


Figure 3. Conceptual framework for least-privilege RBAC optimization and benchmarking.

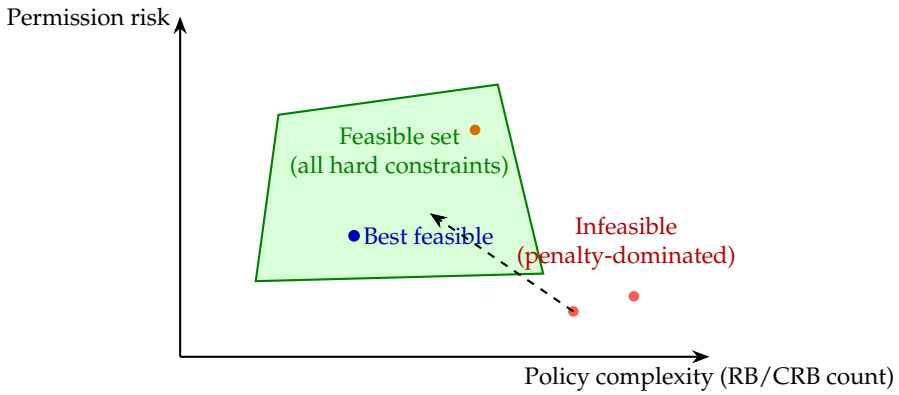


Figure 4. Conceptual view of the search landscape: penalties steer the optimizer far from infeasible configurations toward the feasible set, where risk and complexity are minimized to acquire the last least-privilege policy.

4. Experimental Setup

4.1. RBAC Benchmark

We inspect a Kubernetes-inspired role-based access control (RBAC) design issue in which the objective is to appoint namespace roles and cluster roles to subjects through namespaces while implementing least privilege. Each individual assignment determines the permissions provided to a subject and incurs (i) a permission-risk cost and (ii) a binding cost. The benchmark contains three subjects (ci-bot, frontend-sa, auditor), two namespaces ({dev, prod), eight namespace roles, and four cluster roles. Each role provided a set of permissions (namespaced or cluster-scoped), and requirements determine which permissions each subject must hold in each namespace. Namespace allowlists, role scoping rules, admin approval for cluster-role bindings, binding budgets, and separation-of-duties constrains. are encoded by hard constrains.

The benchmark is thus a controlled snapshot benchmark. It is intended to compare optimizer behavior on a shared constrained RBAC instance, not to claim that a single offline run is enough for the entire lifecycle of a production Kubernetes cluster. The practical scenario illustrated through the benchmark is a limited but realistic Kubernetes application team: a CI/CD service account launches only in the development namespace, an application service account functions only the production frontend workload, and an auditor acquires read-only inspection authority. This makes the practical purpose quantitative: the optimizer must hold all needed permissions while reducing unnecessary grants, risky cluster-wide authority, and binding complexity.

The continuous decision vector has $n_X = 3 \times 2 \times 12 = 72$ variables for namespaced bindings and $n_Y = 3 \times 4 = 12$ variables for cluster-role bindings (total dimension $D = 84$). Each variable contained in $[0, 1]$ and is thresholded at 0.5 to get binary binding decisions. A candidate is evaluated using a penalised objective

$$f = f_{\text{base}} + f_{\text{pen}},$$

where f_{base} combines permission risk and binding cost, and f_{pen} is a non-negative penalty that increases with violated hard constraints. In addition to f, we report least-privilege quality metrics that quantify how well needed permissions are covered while avoiding unnecessary privileges.

4.2. Search Algorithms

We compare ten metaheuristic and baseline search methods implemented in Python 3.14 using a common interface:

- Particle Swarm Optimisation (PSO) [31].
- Grey Wolf Optimiser (GWO) [32].
- Whale Optimisation Algorithm (WOA) [33].
- Sine–Cosine Algorithm (SCA) [34].
- Differential Evolution (DE) [35].
- Genetic Algorithm (GA) [36].
- Bat Algorithm (BA) [37].
- Simulated Annealing (SA) [38].
- Random Search (RS) [39].
- Hill Climbing (HC) [40].

Population-based approaches utilized a population capacity of 20 agents and were operated for 300 iterations. Each optimiser was repeated for 5 distinct runs. Run seeds were fixed deterministically as

$$\text{seed}(o, r) = 12{,}345 + 10{,}000\,(o + 1) + r,$$

where o is the optimiser index and $r \in \{1, \ldots, 5\}$ is the run id, to ensure reproducibility.

4.3. Evaluation Metrics

For each iteration, we test feasibility (whether all hard constraints are satisfied), the final penalised objective f, and a decomposition into base objective f_{base} and penalty f_{pen}. Least-privilege quality is measured using standard information-retrieval metrics [41,42]: Recall $= \frac{\text{TP}}{\text{TP+FN}}$, Precision $= \frac{\text{TP}}{\text{TP+FP}}$, and F1 is their harmonic mean. Here, μ_{F1} denotes the arithmetic mean of F1 over the five independent runs in Table 3, whereas the F1 column in Table 4 refers to the single best run of each optimizer. The two values therefore attend to different questions: average reliability across runs versus most desirable observed policy quality. We also address the over-privilege percentage (extra permissions divided by all granted permissions), the count of role bindings (RB) and cluster-role bindings (CRB), the overall granted-permission risk, and the incremental risk beyond the minimum baseline risk imposed by requirements. All addressed indicators are utilized in the evaluation: feasibility and penalty quantify constraint satisfaction; fitness and base objective quantify optimization quality; precision, recall, F1, and over-privilege quantify least-privilege security; RB/CRB counts quantify administrative complexity; and excess risk quantifies security improvement beyond the minimum required-permission baseline.

Table 3. Summary statistics for ten optimisers over 5 runs and 300 iterations: feasible rate, mean (μ) and standard deviation (σ) of final fitness, median fitness, and mean F1. Lower fitness is better.

Optimiser	Feasible Rate	μ_{fit}	σ_{fit}	Median Fitness	μ_{F1}
PSO	0.00	1.27×10^6	2.60×10^5	1.31×10^6	0.597
GWO	0.40	8.20×10^4	8.44×10^4	1.00×10^5	0.859
WOA	0.00	8.20×10^5	5.15×10^5	6.60×10^5	0.666
SCA	0.40	1.20×10^5	1.10×10^5	2.00×10^5	0.830
DE	0.40	8.40×10^4	8.53×10^4	1.00×10^5	0.814
GA	0.00	7.80×10^5	3.52×10^5	5.70×10^5	0.629
BA	0.00	2.75×10^6	2.71×10^5	2.85×10^6	0.513
SA	0.00	9.64×10^5	1.85×10^5	8.70×10^5	0.652
RS	0.00	1.92×10^6	1.03×10^5	1.93×10^6	0.546
HC	0.00	6.62×10^5	6.94×10^4	7.00×10^5	0.682

Table 4. Results metrics for the best run of each optimiser: feasibility, fitness, recall, precision, F1, and over-privilege percentage.

Optimiser	Feasible	Fitness	Recall	Precision	F1	Over-Priv.
PSO	0	8.90×10^5	1.000	0.447	0.618	0.553
GWO	1	17.05	1.000	0.739	0.850	0.261
WOA	0	3.10×10^5	1.000	0.654	0.791	0.346
SCA	1	17.05	1.000	0.739	0.850	0.261
DE	1	23.20	1.000	0.708	0.829	0.292
GA	0	4.90×10^5	1.000	0.486	0.654	0.514
BA	0	2.40×10^6	1.000	0.327	0.493	0.673
SA	0	8.10×10^5	1.000	0.515	0.680	0.485
RS	0	1.77×10^6	1.000	0.378	0.548	0.622
HC	0	5.60×10^5	1.000	0.500	0.667	0.500

5. Results

Table 3 reports performance over 5 runs. Only GWO, SCA, and DE satisfied feasible solutions (feasible rate = 0.40 each); all outstanding approaches were ineffective to comply with all hard conditions within the same budget. Among the feasible-capable methods, GWO achieved the highest mean F1 (0.859), while GWO and DE achieved the lowest mean fitness values ($\approx 8.2 \times 10^4$ and 8.4×10^4, respectively), reflecting considerably smaller penalties than the remaining approaches. Figures 5 and 6 additionally illustrate that GWO/SCA/DE minimize the penalised objective quicker and with considerably smaller dispersion than BA/RS/PSO. How ever, the mean convergence curves (best-so-far fitness) averaged over five runs is shown in Figure 5 and the Distribution of final fitness values across optimiser isshown in Figure 6.

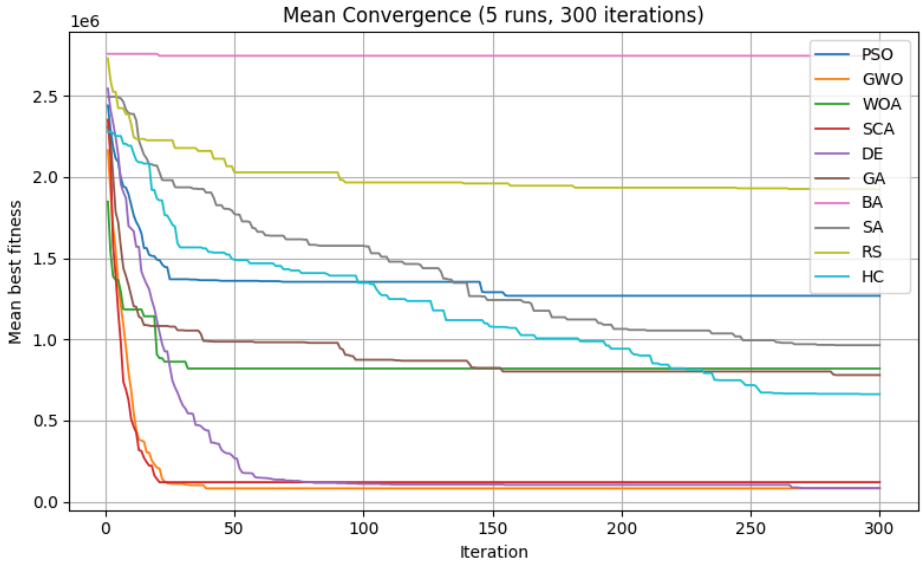


Figure 5. Mean convergence curves (best-so-far fitness) averaged over five runs. GWO, SCA, and DE reduce penalties much faster than the remaining methods.

To determine best-case behavior, Table 4 provides the best (lowest-fitness) run of each optimizer. All runs obtained Recall = 1.0, demonstrating that needed permissions were always covered; consequently, differences in F1 are driven primarily by precision (i.e., reducing unnecessary permissions). GWO and SCA attained the best F1 (0.850) with the lowest over-privilege ratio (0.261), whereas BA and RS created highly over-privileged configurations (over-privilege ratio ≥ 0.622) and the lowest F1 scores.

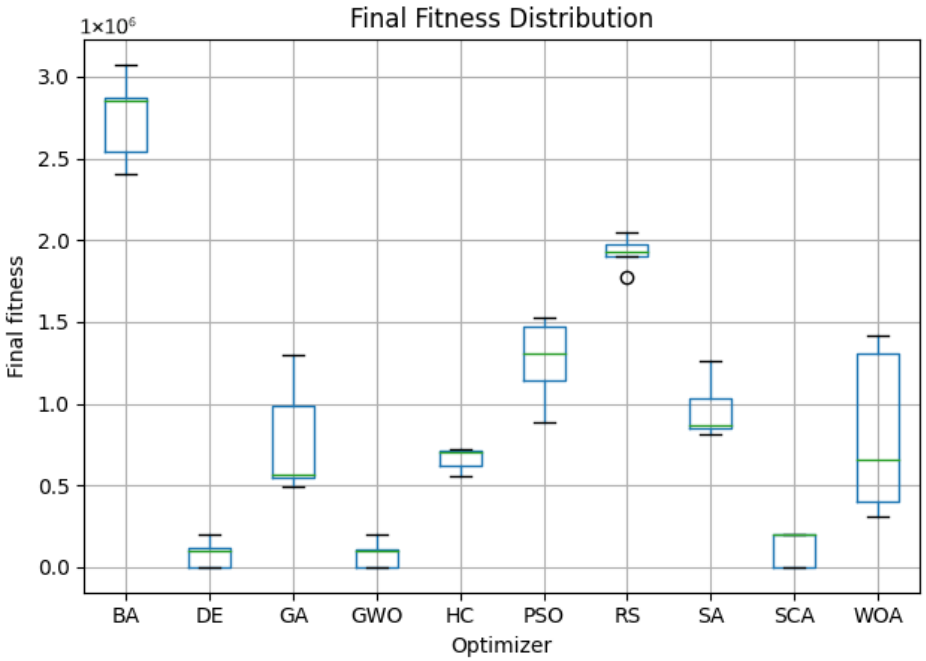


Figure 6. Distribution of final fitness values across optimisers. GWO/SCA/DE are concentrated at substantially lower fitness values, whereas BA/RS remain at large penalty levels.

Structural components of the best solutions are illustrated in Table 5. GWO and SCA produce extremely compact policies (RB = 3, CRB = 1) with negligible excess risk (2.5), whereas BA/RS create bloated policies (RB = 34 and 31, CRB = 4) and considerable excess risk (>250). Apart from the best-run analysis, Table 6 reports mean penalty, mean base objective, and average policy size over all runs. Across all optimisers, the penalty share exceeds 0.999, confirming that constraint compliance is the major challenge in this benchmark. Lastly, Table 7 quantifies least-privilege behavior via the mean number of provided permissions and extras: GWO grants on average 22.6 permissions (only 5.6 extra), while BA grants 49.4 permissions (32.4 extra).

The indicators in Tables 3–7 collectively illustrate the practical security enhancement. Relative to the weakest best-run policy (BA), the best GWO/SCA policy minimizes excess risk from 323.20 to 2.50, a 99.2% reduction, and minimizes the quantity of bindings from 38 total bindings (RB = 34, CRB = 4) to 4 total bindings (RB = 3, CRB = 1). Relative to the average BA outcome in Table 7, GWO minimizes extra permissions from 32.4 to 5.6, an 82.7% reduction, while raising mean F1 from 0.513 to 0.859. These numbers quantify the security advantage of the optimized least-privilege configuration: lesser unnecessary permissions, considerably decrease excess risk, and a more auditable RBAC structure.

Table 5. Structural and risk characteristics of the best solutions: number of role bindings (RB), cluster-role bindings (CRB), and excess risk above the required-risk baseline.

Optimiser	RB	CRB	Excess Risk
PSO	15	2	216.50
GWO	3	1	2.50
WOA	6	1	40.40
SCA	3	1	2.50
DE	6	1	8.50
GA	14	3	176.50
BA	34	4	323.20
SA	16	3	128.40

Table 5. *Cont.*

Optimiser	RB	CRB	Excess Risk
RS	31	4	257.20
HC	11	2	142.50

Table 6. Mean objective decomposition and policy size over all runs: mean base objective (f_{base}), mean penalty (f_{pen}), penalty share f_{pen}/f, and average RB/CRB and additional risk.

Optimiser	Feas. Rate	f_{base}	f_{pen}	f_{pen}/f	Over-Priv.	RB	CRB	Excess Risk
GWO	0.40	24.91	8.20×10^4	0.9997	0.246	2.6	1.2	10.34
DE	0.40	41.92	8.40×10^4	0.9995	0.314	6.0	1.0	27.22
SCA	0.40	42.25	1.20×10^5	0.9996	0.291	3.0	1.0	27.70
HC	0.00	116.20	6.62×10^5	0.9998	0.481	15.2	2.6	100.72
GA	0.00	156.49	7.80×10^5	0.9998	0.540	13.0	3.0	141.04
WOA	0.00	152.53	8.20×10^5	0.9998	0.492	11.4	1.8	137.40
SA	0.00	160.47	9.64×10^5	0.9998	0.515	19.8	2.6	144.76
PSO	0.00	203.81	1.27×10^6	0.9998	0.574	18.2	2.4	188.22
RS	0.00	246.63	1.92×10^6	0.9999	0.624	27.0	5.4	230.00
BA	0.00	315.17	2.75×10^6	0.9999	0.655	37.0	5.6	298.00

Table 7. Least-privilege behaviour over all runs: mean precision, mean F1, mean over-privilege percentage, and mean numbers of granted and additional privileges (Needed permissions remain fixed at 17).

Optimiser	Precision	F1	Over-Priv.	Granted Perms	Extra Perms
GWO	0.754	0.859	0.246	22.6	5.6
SCA	0.709	0.830	0.291	24.0	7.0
DE	0.686	0.814	0.314	24.8	7.8
HC	0.519	0.682	0.481	33.0	16.0
WOA	0.508	0.666	0.492	35.2	18.2
SA	0.485	0.652	0.515	35.4	18.4
GA	0.460	0.629	0.540	37.4	20.4
PSO	0.426	0.597	0.574	40.0	23.0
RS	0.376	0.546	0.624	45.4	28.4
BA	0.345	0.513	0.655	49.4	32.4

6. Discussion

6.1. Constraint Satisfaction Dominates the Objective

The main finding is that the penalised objective is greatly driven by constraint violations instead of by the base risk-and-cost term. Table 6 shows that, for every optimiser, the average penalty share f_{pen}/f exceeds 0.999. This way the substantial gap between the feasible-capable methods (GWO/SCA/DE) and the rest is made clear: when an algorithm isn't able to enter the feasible region, even a near-optimal f_{base} (typically on the order of 10^2) is negligible compared to penalties that rapidly reach 10^6. As a consequence, performance on this benchmark is mostly determined by the ability to navigate a complex feasible set defined by multiple interacting RBAC constraints (allowlists, scoping, approvals, budgets and separation-of-duties).

6.2. Why GWO/SCA/DE Outperform the Others

GWO, SCA, and DE consistently get lower fitness values because their update mechanisms maintain population diversity while still achieving the best solutions (so far). GWO updates candidate solutions to a small set of leaders (alpha/beta/delta) [32], which helps

to stabilize progress once a near-feasible region is found. SCA uses oscillatory sine–cosine steps that naturally shrink over time [34], which supports rough exploration early and more in-depth adjustments later. DE uses differential mutation and recombination [35], which helps it pass over continuous landscapes and escape local traps. Those properties are reflected in Figure 5: the mean best-so-far curves of GWO/SCA/DE drop quickly, showing frequent reductions in penalties and eventually discovery of near-feasible or feasible assignments.

In comparison, PSO and GA, while effective in many continuous settings [31,36], tend to converge to configurations that still breach at least one hard constraint with the limited budget. Similarly, WOA [33] sometimes finds lower-penalty regions (Table 3) but shows higher variability and more stagnation. Single-solution and baseline methods (SA/HC/RS) don't have strong population-level information sharing, which reduces the chances of simultaneously satisfying multiple coupled limitations.

6.3. Least-Privilege Quality: Recall Saturates, Precision Differentiates

Across all runs, Recall = 1.0 (Table 4), that shows that needed permissions were always given. This behavior suggests that the benchmark/evaluator greatly enforces requirement coverage, if by creating candidate bindings or by using penalties on missing requirements. The result is that the least-privilege quality is mainly determined by precisionand the over-privilege ratio [41,42]. Table 7 shows a clear order which is: GWO has the highest mean precision (0.754) and the lowest mean over-privilege ratio (0.246), followed by SCA and DE. The weaker methods give many extra permissions (BA grants on average 32.4 extra permissions), producing low precision and low F1. This finds that even when required access is achieved, different optimizers shift toward over-permissioned solutions unless they were guided by mechanisms that specifically decrease superfluous grants.

6.4. Policy Compactness and Excess Risk

Not only is least privilege in RBAC about permission numbers but also about the structure of the producing policy. Compact policies are easier to audit, maintain, and reason about. The best configurations produced by GWO and SCA need only three role bindings and one cluster-role binding (Table 5), with neglected excess risk. The RB = 3, CRB = 1 policy is sufficient for the studied benchmark, because the benchmark has only three subjects, two namespaces, and a limited set of required namespace and cluster permissions. It should be interpreted as a fixed optimized policy for the evaluated Kubernetes-style instance instead of as a recommended universal role count for arbitrary systems. Larger groups with more service accounts, namespaces, custom resources and application teams will obviously require a larger optimized binding set. In comparison, BA/RS construct many more bindings (tens of RB/CRB), which correlates with increased over-privilege and elevated excess risk. We anticipate this coupling, as more bindings usually means broader permission unions which increases the risk and makes it harder to satisfy separation-of-duties and binding-budget constraints at the same time.

6.5. Feasible-Only Comparison Among the Top Methods

Because only GWO/SCA/DE generated feasible solutions, comparing their feasible runs in isolation is very useful. Across feasible runs, GWO gives the best least-privilege profile: mean feasible fitness 19.85, mean precision 0.756, mean over-privilege 0.244 and mean excess risk 5.30. These feasible-run averages are calculated from the subset of successful runs behind Tables 3–7; they are not separate table entries because Tables 3, 6 and 7 report all-run averages, while Table 4 reports the best run. We only clarified this distinction to avoid mixing all-run, best-run and feasible-only summaries. SCA is very close (mean feasible fitness 20.00 and excess risk 5.45), although DE performs close to this but usually creates slightly larger policies and higher excess risk on feasible runs (mean excess risk

14.00). These results indicate that multiple mechanisms (leader-based guidance in GWO and oscillatory adjustment in SCA) can reliably converge to very compact and low-risk RBAC designs when feasibility is founded.

6.6. Dynamic RBAC Snapshots and Adaptive Optimization

The current formulation uses a fixed RBAC snapshot, which is an intentional simplification for mathematical clarity and fair benchmarking. Real Kubernetes clusters are more dynamic: services are deployed and scaled continuously, service accounts can be created by controllers, custom resources and API groups are improved and audit-log access patterns can shift as applications change. Consequently, the proposed method should be viewed as a batch hardening component, instead of a complete long-term RBAC governance system on it's own.

A practical extension is to embed the optimizer in an adaptive control loop. At optimization epoch t, the inputs can be rebuilt from the current manifests, live RBAC objects, audit logs and policy constraints. The objective can then include an explicit change-cost term so that the optimizer chooses safe incremental updates instead of disruptive policy rewrites:

$$f_t(x_t, y_t) = J_t(x_t, y_t) + \lambda_\Delta D\big((x_t, y_t), (x_{t-1}, y_{t-1})\big) + \sum_i \rho_i V_{i,t}(x_t, y_t), \tag{19}$$

where $D(\cdot)$ measures the number or severity of RBAC edits relative to the previously approved policy. Candidate updates can then be checked and validated by Kubernetes dry-run checks, policy simulation, canary rollout, admission-control rules and rollback monitoring before being applied cluster-wide. This extension will turn the current snapshot optimizer into a continuous least-privilege maintenance process.

6.7. Risk-Weight Calibration and Subjectivity

One more limitation is the manually assigned permission-risk weights used in the objective. They make the model interpretable and easy to adapt, but they can also introduce bias because a high-risk action in one cluster can be normal in another, and security teams might disagree about the relative severity of verbs, resources, namespaces and subjects. If the weights are not calibrated well, the optimizer might still satisfy all hard constraints while prioritizing a risk profile that is not the same as the organization's actual threat model.

In practice, risk weights should be viewed as governance parameters instead of arbitrary constants. They can be calibrated using expert elicitation, compliance requirements, resource sensitivity labels, namespace criticality, incident history, exposure levels and empirical audit-log usage. We should also run sensitivity analysis by repeating optimization under multiple plausible weight settings and checking if the same RBAC design remains preferred or not. This way we can reduce subjectivity and make the reported least-privilege gains more robust.

6.8. Statistical Evidence and Limitations

To evaluate whether differences persist across runs, we ranked optimisers per run (lower fitness is better, higher F1 is better) and applied the Friedman test among the five runs. The test shows significant differences among optimisers for both fitness ($p = 7.89 \times 10^{-6}$) and F1 ($p = 1.18 \times 10^{-5}$). However, with only 5 runs, post-hoc pairwise tests have limited power: Wilcoxon signed-rank comparisons with Holm correction does not reach 0.05 significance even when the top method dominates all run-wise comparisons. As a result, the best evidence in this study is the effect magnitude (orders-of-magnitude penalty reductions) and the consistent qualitative patterns over Tables 3–7 and Figures 5–7.

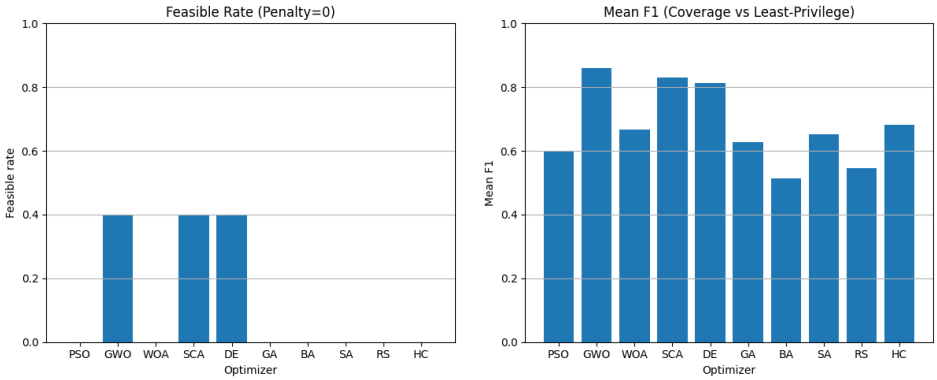


Figure 7. Feasible rate (**left**) and mean F1 score (**right**). Only GWO, SCA, and DE produced feasible solutions within the evaluated budget.

Overall the findings show that (i) feasibility is the primary bottleneck, (ii) GWO/SCA/DE are significantly better at reducing penalties and generating compact policies and (iii) between feasible solutions, GWO achieves the best precision/over-privilege trade-off. Future research should focus on improving constraint-handling (e.g., repair operators and feasibility-preserving moves) to increase the feasible rate beyond 40%, and increase reliability for practical deployments.

7. Conclusions

This paper frames least-privilege RBAC hardening for Kubernetes-oriented cloud container systems as a constrained optimization problem that reduces permission risk and operational complexity under practical deployment requirements. The model specifically captures RBAC semantics for namespace and cluster-scoped permissions and enables real-world constrains like namespace allowlists, role scoping rules, administrative gating for cluster-wide bindings, binding budgets and separation-of-duty limitations defined across capability classes. To allow for optimizer-agnostic search, all the constraints are combined into a unified penalty-based fitness function, allowing for black-box metaheuristics to be benchmarked on a share evaluation framework.

Experimental results on a Kubernetes-inspired benchmark show constraint satisfaction is the key difficulty, when an optimizer does not attain feasibility, penalties dominate the improvements in the base risk-and-cost objective. Only a subset of the evaluated methods reliably founded feasible solutions within the same budget, and through feasible configurations the best policies were both compact (few RB/CRB) and close to least privilege (high precision and low over-privilege), with least excess risk beyond the requirement baseline, which confirms the use of metaheuristic optimization as an effective element in automated RBAC minimization processes for cloud container security.

Practically speaking the best optimized policy preserves all required permissions while reducing unneeded grants and excess risk by large margins in comparison to weaker search baselines. Therefore the improvement is algorithmic and also security-oriented and the resulting policy has less bindings to audit, less extra permissions available to a compromised workload and less cluster-wide authority than over-privileged alternatives.

Future Work

Many directions can improve deployability and scalability:

- **Feasibility-first and repair operators:** incorporate feasibility-preserving and constraint-repair steps (e.g., binding pruning, SoD-aware swaps) to increase feasible rates more than the observed levels.

- **Adaptive and structured penalties:** dynamically tune penalty weights or use staged penalties (requirements → validity → budgets → SoD) to better guide search at highly constrained instances.
- **Multi-objective optimization:** explicitly compute Pareto fronts for (risk, complexity, change-cost) instead of collapsing everything into a single scalar objective.
- **Scaling to real clusters:** evaluate larger, noisy datasets derived from audit logs and production RBAC states, and study robustness under workload drift.
- **Safer deployment loops:** integrate canary evaluation, rollback policies and continuous verification, so that privilege tightening does not disrupt legitimate workloads.
- **Adaptive RBAC maintenance:** extend snapshot formulation into a recurring optimization process that refreshes requirements from audit logs and manifests, minimizes change cost and validates candidate policies before rollout.
- **Risk-weight calibration:** switch pure manual risk settings for documented expert elicitation, sensitivity analysis and data-assisted calibration based on resource sensitivity, namespace criticality and observed access patterns.

Author Contributions: Conceptualization, W.K.A., S.S., H.K., and E.M.I.; methodology, W.K.A., H.K., E.M.I., and S.S.; software, W.K.A., S.S., and L.A.; validation, N.A.K., H.N.F., and L.A.; formal analysis, W.K.A., L.A., S.S. and S.F.; investigation, W.K.A., E.M.I. and H.K.; resources, E.M.I. and H.K.; data curation, S.S.; writing—original draft preparation, W.K.A.; writing—review and editing, N.A.K., E.M.I., H.K., and S.S.; visualization, N.A.K. and H.N.F.; supervision, W.K.A.; project administration, W.K.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Saltzer, J.H.; Schroeder, M.D. The protection of information in computer systems. *Proc. IEEE* **1975**, *63*, 1278–1308. [CrossRef]
2. Sandhu, R.S. Role-based access control. *Adv. Comput.* **1998**, *46*, 237–286.
3. Ferraiolo, D.F.; Sandhu, R.; Gavrila, S.; Kuhn, D.R.; Chandramouli, R. Proposed NIST standard for role-based access control. *ACM Trans. Inf. Syst. Secur.* **2001**, *4*, 224–274. [CrossRef]
4. Sanders, M.W.; Yue, C. Minimizing privilege assignment errors in cloud services. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*; Association for Computing Machinery: New York, NY, USA, 2018; pp. 2–12.
5. Rastogi, V.; Davidson, D.; Carli, L.D.; Jha, S.; McDaniel, P. Towards least privilege containers with cimplifier. *arXiv* **2016**, arXiv:1602.08410 [CrossRef]
6. Vaidya, J.; Atluri, V.; Guo, Q. The role mining problem: Finding a minimal descriptive set of roles. In *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies*; Association for Computing Machinery: New York, NY, USA, 2007; pp. 175–184.
7. Ene, A.; Horne, W.; Milosavljevic, N.; Rao, P.; Schreiber, R.; Tarjan, R.E. Fast exact and heuristic methods for role minimization problems. In *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies*; Association for Computing Machinery: New York, NY, USA, 2008; pp. 1–10.
8. Karim, N.; Khashan, O.; Abdulraheem, W.; Alazab, M.; Kanaker, H.; Farfoura, M.; Alshinwan, M. Performance comparison of Hyper-V and KVM for cryptographic tasks in cloud computing. *Comput. Mater. Contin.* **2024**, *78*, 2023. [CrossRef]
9. D'Antoni, L.; Ding, S.; Goel, A.; Ramesh, M.; Rungta, N.; Sung, C. Automatically reducing privilege for access control policies. *Proc. Acm Program. Lang.* **2024**, *8*, 763–790. [CrossRef]
10. Jarkas, O.; Ko, R.K.L.; Dong, N.; Mahmud, M.R. A container security survey: Exploits, attacks, and defenses. *ACM Comput. Surv.* **2025**, *57*, 170:1–170:36. [CrossRef]
11. Lu, H.; Vaidya, J.; Atluri, V. Optimal boolean matrix decomposition: Application to role engineering. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*; IEEE: New York, NY, USA, 2008; pp. 297–306.

12. Molloy, I.; Chen, H.; Li, T.; Wang, Q.; Li, N.; Bertino, E.; Calo, S.; Lobo, J. Mining roles with semantic meanings. In *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies*; Association for Computing Machinery: New York, NY, USA, 2008; pp. 21–30.
13. Xu, Z.; Stoller, S.D. Algorithms for mining meaningful roles. In *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies*; Association for Computing Machinery: New York, NY, USA, 2012; pp. 57–66.
14. Mitra, B.; Sural, S.; Vaidya, J.; Atluri, V. A survey of role mining. *ACM Comput. Surv.* **2016**, *48*, 1–37. [CrossRef]
15. Vaidya, J.; Atluri, V.; Guo, Q.; Adam, N. Migrating to optimal RBAC with minimal perturbation. In *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies*; Association for Computing Machinery: New York, NY, USA, 2008; pp. 11–20.
16. Ma, X.; Li, R.; Lu, Z.; Lu, J.; Dong, M. Specifying and enforcing the principle of least privilege in role-based access control. *Concurr. Comput. Pract. Exp.* **2011**, *23*, 1313–1331. [CrossRef]
17. Huang, H.; Shang, F.; Liu, J.; Du, H. Handling least privilege problem and role mining in RBAC. *J. Comb. Optim.* **2015**, *30*, 63–86. [CrossRef]
18. Sanders, M.W.; Yue, C. Mining least privilege attribute based access control policies. In *Proceedings of the 35th Annual Computer Security Applications Conference*; Association for Computing Machinery: New York, NY, USA, 2019; pp. 404–416.
19. Rao, K.R.; Nayak, A.; Ray, I.G.; Rahulamathavan, Y.; Rajarajan, M. Role recommender-RBAC: Optimizing user-role assignments in RBAC. *Comput. Commun.* **2021**, *166*, 140–153. [CrossRef]
20. Tak, B.; Isci, C.; Duri, S.; Bila, N.; Nadgowda, S.; Doran, J. Understanding security implications of using containers in the cloud. In *Proceedings of the 2017 USENIX Annual Technical Conference (USENIX ATC 17)*; USENIX Association: Berkeley, CA, USA, 2017; pp. 313–319.
21. Torkura, K.A.; Sukmana, M.I.H.; Cheng, F.; Meinel, C. CAVAS: Neutralizing application and container security vulnerabilities in the cloud native era. In *Proceedings of the International Conference on Security and Privacy in Communication Systems*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 471–490.
22. Sun, J.; Wu, C.; Ye, J. Blockchain-based automated container cloud security enhancement system. In *Proceedings of the 2020 IEEE International Conference on Smart Cloud (SmartCloud)*; IEEE: New Tork, NY, USA, 2020; pp. 1–6.
23. Gao, X.; Gu, Z.; Kayaalp, M.; Pendarakis, D.; Wang, H. Containerleaks: Emerging security threats of information leakages in container clouds. In *Proceedings of the 2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*; IEEE: New Tork, NY, USA, 2017; pp. 237–248.
24. Zhu, H.; Gehrmann, C.; Roth, P. Access security policy generation for containers as a cloud service. *SN Comput. Sci.* **2023**, *4*, 748. [CrossRef]
25. Li, Y.; Hu, H.; Liu, W.; Yang, X. An optimal active defensive security framework for the container-based cloud with deep reinforcement learning. *Electronics* **2023**, *12*, 1598. [CrossRef]
26. Yu, Z.; Ye, L.; Zhang, H.; Zhan, D.; Su, S.; Tian, Z. A container-oriented virtual-machine-introspection-based security monitor to secure containers in cloud computing. In *Proceedings of the International Conference on Artificial Intelligence and Security*; Springer: Berlin/Heidelberg, Germany, 2021; pp. 102–111.
27. Gao, X.; Steenkamer, B.; Gu, Z.; Kayaalp, M.; Pendarakis, D.; Wang, H. A study on the security implications of information leakages in container clouds. *IEEE Trans. Dependable Secur. Comput.* **2018**, *18*, 174–191. [CrossRef]
28. Eldjou, A.; Kitouni, I.; Benmounah, Z.; Bennacer, S. Enhancing cloud native security: A knowledge graph approach for securing container runtimes. *Clust. Comput.* **2025**, *28*, 777. [CrossRef]
29. Nascimento, B.; Santos, R.; Henriques, J.; Bernardo, M.V.; Caldeira, F. Availability, scalability, and security in the migration from container-based to cloud-native applications. *Computers* **2024**, *13*, 192. [CrossRef]
30. Timofte, E.M.; Balan, A.L.; Iftime, T. AI driven adaptive security mesh: Cloud container protection for dynamic threat landscapes. In *Proceedings of the 2024 International Conference on Development and Application Systems (DAS)*; IEEE: New Tork, NY, USA, 2024; pp. 71–77.
31. Kennedy, J.; Eberhart, R. Particle swarm optimization. In *Proceedings of ICNN'95-International Conference on Neural Networks*; IEEE: New Tork, NY, USA, 1995; Volume 4, pp. 1942–1948.
32. Mirjalili, S.; Mirjalili, S.M.; Lewis, A. Grey wolf optimizer. *Adv. Eng. Softw.* **2014**, *69*, 46–61. [CrossRef]
33. Mirjalili, S.; Lewis, A. The whale optimization algorithm. *Adv. Eng. Softw.* **2016**, *95*, 51–67. [CrossRef]
34. Mirjalili, S. SCA: A sine cosine algorithm for solving optimization problems. *Knowl.-Based Syst.* **2016**, *96*, 120–133. [CrossRef]
35. Alkhatib, A.; Shaheen, A.; Albustanji, R.N. *A Comparative Analysis of Cloud Computing Services: AWS, Azure, and GCP*; University of Bahrain: Sakhir, Bahrain, 2024.
36. Holland, J.H. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*; MIT Press: Cambridge, MA, USA, 1992.
37. Yang, X.-S. A new metaheuristic bat-inspired algorithm. In *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 65–74.
38. Kirkpatrick, S.; Gelatt, C.D., Jr.; Vecchi, M.P. Optimization by simulated annealing. *Science* **1983**, *220*, 671–680. [CrossRef]

39. Bergstra, J.; Bengio, Y. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.* **2012**, *13*, 281–305.
40. Al Khaldy, M.; Shaheen, A.; Alzyadat, W.; Alhroob, A. An efficient primary indexing method with sibling pointers for large-scale database systems. *Eng. Technol. Appl. Sci. Res.* **2025**, *15*, 24691–24697. [CrossRef]
41. Manning, C.D.; Raghavan, P.; Schütze, H. *Introduction to Information Retrieval*; Cambridge University Press: Cambridge, UK, 2008.
42. Powers, D.M.W. Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation. *arXiv* **2020**, arXiv:2010.16061. [CrossRef]

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