

**Ganna Grynkevych**

DSc, Associate Professor,

Professor of the Department of Information and Cyber Security named after Professor Volodymyr Buriachok,
Borys Grinchenko Kyiv Metropolitan University, Kyiv, Ukraine

ORCID: 0000-0003-1922-5165

ggrynkevych@ukr.net

Volodymyr Vasylenko

PhD, Associate Professor,

Associate Professor of the Department of Computer Science,

State University of Information and Communication Technologies, Kyiv, Ukraine

ORCID: 0000-0001-8465-6178

oknelisavvova172@gmail.com

**A USAGE-AWARE METHOD FOR REPAIRING PRIVILEGE ESCALATION PATHS IN AWS
IDENTITY AND ACCESS MANAGEMENT**

Abstract. Misconfigured identity policies in Amazon Web Services (AWS) let a low-privileged user or role obtain administrator rights through chains of individually harmless permissions such as `iam:PassRole` or `iam:CreateAccessKey`. Existing repair methods choose the fix by the number of policy changes, so a “minimal” repair can break deployments, service roles and people who use the permission for legitimate work. This paper proposes a usage-aware method for repairing privilege escalation paths in AWS Identity and Access Management (IAM). The account is modeled as a permission graph with monotone escalation rules, and usage evidence from access logs defines operational impact as the weighted number of observed (principal, action, resource) uses that a repair would break. Repair is solved as an exact counterexample-guided CP-SAT optimization that either minimizes impact under full safety or maximizes removed risk under a change budget; under stated monotonicity assumptions the result is optimal within the repair vocabulary. On 50 synthetic organizations the method removed all escalations while breaking 78% fewer legitimate uses than a minimum-size repair (32 versus 144; Wilcoxon test, $p < 10^{-6}$), at the cost of a median of 6.5 instead of 3 edited policy statements. Hiding `iam:PassRole` from the evidence, as AWS last-accessed data does, raised impact from 1.21 to 2.18, and a conservative prior for unobservable actions brought it back to 1.21. With four changes the budgeted variant removed 96.7% of escalation risk. On a public export of the IAM Vulnerable account the model detected 50 of 58 in-scope escalation scenarios, and usage-aware repair reduced impact by 65.7% relative to a minimum-size repair.

Keywords: cloud security; AWS IAM; privilege escalation; least privilege; access control repair; MaxSAT; constraint programming; usage logs.

INTRODUCTION

In AWS every API call is authorized against identity policies attached to users, groups and roles, and against resource policies such as role trust policies. Large accounts accumulate thousands of such statements, written by different teams over years, and some of them are dangerous in combination. A developer group that may create access keys “for service users” can, if the resource is a wildcard, create a key for an administrator. A deployment role that may pass “its” execution role to AWS Lambda can pass an administrator role instead and run code with that role’s rights. The 2019 Capital One breach, in which a role reachable from a compromised compute instance exposed personal data of about 100 million credit card customers and applicants, shows how far one role can reach [1, 2]. Rhino Security Labs documented 21 escalation methods in 2018 [3], and later catalogs add more [4].

Several tools detect such paths. PMapper builds a graph of which principal can act as which [5]; a model checker [6] and a hybrid fixed-point and learning analysis [7] search multi-step attacks. Repair has received less attention. The two closest systems are IAM-Deescalate [8], which adds explicit deny statements to each risky principal, and IAMPERE [2], which encodes repair as MaxSAT over a permission flow graph and finds an approximately minimum patch. Both count changes; neither looks at whether the permission being revoked is in



use.

Broad grants usually cover a narrow legitimate use. A CI role passes its own application role to Lambda every day and never the administrator role; three developers create keys only for their team's service user. A repair that deletes the grant closes the path and stops the pipeline, which is probably one reason why reported grants are often left in place. AWS provides data that could avoid such breakage, namely last-accessed information and CloudTrail logs [9], but we found no repair method that uses it to choose between alternative cuts of an escalation path.

Purpose of the article and contributions. The aim of this study is to reduce the operational damage caused by repairing privilege escalation in AWS IAM, without weakening the security guarantee, by making the repair objective depend on observed permission usage. To reach this aim we:

1. define a permission model of an AWS account with monotone escalation rules, a witness-based safety check and a repair vocabulary that includes grant removal, membership and trust edits and principal-scoped deny statements;
2. define operational impact through legitimate routes of observed (principal, action, resource) usage pairs, including actions that AWS does not record;
3. formulate two exact problems, usage-aware safe repair and budgeted risk reduction, and solve them with a counterexample-guided CP-SAT procedure whose result is optimal within the vocabulary under stated assumptions;
4. evaluate the method on synthetic organizations (full repair, sensitivity to evidence and to the usage model, change budget, scaling to about 1,000 principals) and on a public export of the IAM Vulnerable account.

Object, subject, novelty and practical significance. The object of the research is access control in public cloud infrastructure; the subject is methods for repairing privilege escalation paths in AWS IAM. The scientific novelty lies in (a) a repair objective that counts broken legitimate usage instead of edited permissions, with an exact linear encoding through route sets, and (b) a budgeted variant that trades residual escalation risk against impact, adapted from minimum-cost network hardening to identity graphs. In practice, a security team can close every path found in a snapshot of the account, keep the permissions that people and pipelines actually use, and decide how many changes fit into one change window.

ANALYSIS OF RECENT RESEARCH AND PUBLICATIONS

Escalation catalogs and detection. Gietzen's catalog [3], the IAM Vulnerable test account by Bishop Fox [10] and Datadog's pathfinding.cloud library [4] are the usual references for AWS escalation methods. PMapper builds a principal graph whose edges mean "can act as" [5], TAC models permission flows and provides the IAMVulGen generator of synthetic misconfigurations [7], and Shevrin and Margalit encode multi-step IAM attacks for a model checker [6]. Liu et al. show that permission chaining is also a risk inside serverless applications [11], and agent-based pipelines flag device misconfigurations in enterprise networks [12]. All of them stop at detection and leave the choice of fix to the operator.

Formal reasoning about AWS policies. The Zelkova engine translates policies into SMT and answers questions such as "can anyone outside the account read this bucket" [13]; Block Public Access builds on it [14]. D'Antoni et al. use access logs to reduce the privileges that a single policy grants [15]. Eiers et al. propose quantitative repair of cloud access control policies [16], Restrictor tightens policies automatically [17], and CloudFix uses language models to propose repairs [18]. These approaches reason about one policy or one resource at a time, whereas escalation is a property of the whole graph: the dangerous combination often spans a group policy, a trust policy and a role policy.

Least privilege from usage. Sanders and Yue generated cloud policies from audit logs and measured over-privilege [19, 20]. Vaidya et al. introduced migration to an optimal RBAC configuration with minimal perturbation of existing assignments [21], which is close in spirit to our impact measure. These methods do not reason about escalation paths.

Repair of escalation paths. IAM-Deescalate builds the PMapper graph and, for each edge from a non-administrator to an administrator, denies on the source principal the smallest set of permissions that breaks it; it remediated 24 of the 31 IAM Vulnerable techniques, and users have to review the plan because the denies may affect workloads [8]. IAMPERE [2] encodes repair as MaxSAT over a reduced permission flow graph with unit weights, so the solver minimizes patch size, and a graph neural network proposes an intermediate patch that the solver refines. On 1,000 synthetic configurations IAMPERE repaired 57.6% within 600 s against 21.6% for IAM-Deescalate [2].

Hardening of attack graphs. Attack graphs [22] underlie minimum-cost network hardening, which removes initial conditions so that a goal becomes unreachable [23, 24]. Guo et al. study budgeted edge blocking in Active Directory attack graphs [25], and Abdurraheem et al. optimize Kubernetes RBAC with risk weights and a budget [26]. In this journal, recent work has addressed adaptive context-based access control [27], zero-trust compliance in enterprise networks [28], AI-assisted incident monitoring and response on open-source tools [29]



and threat modeling of offline mesh messaging applications [30]; none of them treats cloud identity graphs.

Table 1 compares the closest approaches with the method proposed here.

Table 1

Comparison with the closest approaches to escalation repair and privilege reduction

Criterion	IAM-Deescalate [8]	IAMPERE [2]	D'Antoni et al. [15]	Network hardening [23, 25]	This work
Object	AWS principals	AWS IAM permission flow graph	One access policy	Attack graph	AWS IAM permission graph
Escalation paths	Yes (PMapper edges)	Yes, transitive	No	Yes (generic)	Yes, transitive
Repair operations	Deny on principal	Revoke assignment or flow	Tighten statements	Remove conditions or edges	Revoke grant, membership or trust; deny on principal
Objective	Fewest denied permissions per escalation edge	Minimum patch size	Allow only observed requests	Minimum cost	Minimum broken legitimate usage
Usage evidence	No	No	Yes (logs)	No	Yes (last accessed, CloudTrail, prior)
Change budget	No	No	No	Yes [25]	Yes
Optimality	Per edge	Approximate (GNN and MaxSAT)	Per policy	Exact or approximate	Exact within vocabulary

Gap. Escalation repair has so far been optimized for patch size or for local minimality per edge, and usage-based tightening has been applied to single policies. What is missing is a method that removes every escalation path in the account graph while minimizing the legitimate usage it breaks, that handles actions for which AWS provides no usage data, and that can stop at a change budget with a known residual risk.

THEORETICAL FOUNDATIONS OF THE STUDY

Threat model. One principal of a single AWS account, a user or a role, is compromised at a time, as in PMapper [5], and the attacker can call every API it is allowed to call. The goal is a target privilege the principal does not hold: administrator equivalence, meaning the ability to write an arbitrary policy to a principal the attacker controls, or a sensitive data privilege such as s3:GetObject on a specific bucket. Out of scope are permission boundaries, service control policies, condition keys, NotResource statements, resource policies other than trust policies, cross-account access and data-plane techniques that rely on existing resources.

Permission model. A configuration is

$$C = (U, G, R, S, \Pi, D, M, T), \quad (1)$$

where U , G and R are users, groups and roles; S are service principals such as `lambda.amazonaws.com`; Π is the set of privileges $\pi = (a, \rho)$ with an action pattern a and a resource pattern ρ ; $D \subseteq (U \cup G \cup R) \times \Pi$ are direct grants obtained by flattening inline and managed policies to statements; $M \subseteq U \times G$ are group memberships; and $T \subseteq R \times (U \cup R \cup S \cup \{\star\})$ are trust relations, $\star$ denoting trust to the account root. Explicit deny statements already present in identity policies are kept as a separate set and evaluated first.

The effective privileges of a user combine its own grants and those of its groups, while a role holds only its own:

$$\text{eff}(e) = D(e) \cup \{\pi \in D(g) : (e, g) \in M\}, \quad e \in U; \quad \text{eff}(r) = D(r), \quad r \in R. \quad (2)$$

A repair changes a set x of elements from the vocabulary

$$X = D \cup M \cup T \cup Y, \quad (3)$$

where removing an element of D , M or T deletes a grant, a membership or a trust entry, and choosing an



element $y = (e, a, \rho) \in Y$ adds a deny statement for action a on resource ρ to principal e . The deny vocabulary contains the concrete resources that occur in escalation witnesses and the wildcard $\rho = *$ for each such action. Grants of the form $(*,*)$ and blanket denies of all actions are protected and never chosen.

Escalation rules. Starting from a compromised principal u , the set $\text{Ctrl}(u)$ of entities whose privileges the attacker can exercise and the flag $\text{Full}(u)$ are the least fixed point of the following rules, where $\text{eff}(\text{Ctrl})$ is the union of the effective privileges of controlled principals:

$$r \in \text{Ctrl} \Leftarrow e \in \text{Ctrl} \wedge [(r, e) \in T \vee ((r, *) \in T \wedge (\text{sts}: \text{AssumeRole}, r) \in \text{eff}(e))], \quad (4)$$

$$v \in \text{Ctrl} \Leftarrow (a, v) \in \text{eff}(\text{Ctrl}), \quad a \in A_{\text{cred}}, \quad (5)$$

$$r \in \text{Ctrl} \Leftarrow (\text{iam}: \text{PassRole}, r) \in \text{eff}(\text{Ctrl}) \wedge (r, s) \in T \wedge L_s \subseteq \text{eff}(\text{Ctrl}), \quad (6)$$

$$\text{Full} \Leftarrow \exists y \in \text{Ctrl}, \quad a \in A_{\text{pw}}(y): (a, y) \in \text{eff}(\text{Ctrl}). \quad (7)$$

Rule (4), R1, is role assumption; a same-account principal named in a trust policy needs no identity grant, whereas trust to the account root does need one [31]. Rule (5), R2, is credential takeover with $A_{\text{cred}} = \{\text{iam}: \text{CreateAccessKey}, \text{iam}: \text{CreateLoginProfile}, \text{iam}: \text{UpdateLoginProfile}\}$. Rule (6), R5, is passing a role to a compute service s that the role trusts; L_s is a set of launch actions for that service, for example $\text{lambda}: \text{CreateFunction}$ with $\text{lambda}: \text{InvokeFunction}$, and launch actions are counted only when granted on all resources. Rule (7), R6, is writing a policy to a controlled entity y with an action from $A_{\text{pw}}(y)$ (Put, Attach, CreatePolicyVersion or SetDefaultPolicyVersion for the type of y); for the two version actions the privilege may also name a customer-managed policy attached to y . Two further rules complete the set: R3, trust rewriting, gives control of r when the attacker holds $\text{iam}: \text{UpdateAssumeRolePolicy}$ and $\text{sts}: \text{AssumeRole}$ on r ; R4, group joining, gives the privileges of g when the attacker holds $\text{iam}: \text{AddUserToGroup}$ on g and controls some user. Groups of a controlled user count as controlled, so writing to one's own group is covered by (7), and controlling a principal that already holds $(*,*)$ sets Full directly, which is a special case of (7).

The privileges reachable from u and the escalations are

$$\text{Priv}(u) = \begin{cases} \Pi, & \text{Full}(u), \\ \text{eff}(\text{Ctrl}(u)), & \text{otherwise,} \end{cases} \quad \text{Esc}(C') = \{(u, \theta): \theta \in \text{Priv}_{C'}(u) \setminus \text{eff}_C(u)\}, \quad (8)$$

for u in the candidate set U_0 (all users and roles that are not administrators and not service-linked) and θ in the target set Θ with risk weights ω_θ . Here $C' = C \setminus x$ is a repaired configuration, and the legitimate holdings $\text{eff}_C(u)$ are always taken from the original configuration C , so that removing a principal's own grant of a target does not turn its later re-acquisition into a different kind of fact. A configuration is safe when $\text{Esc} = \emptyset$.

Witnesses and monotonicity. When the closure derives a fact it records a support set, the elements whose presence the derivation used, including for every exercised privilege the deny element that would block it. For an escalation (u, θ) the support of the final fact is a witness $W \subseteq X$: in the configuration restricted to W the escalation still occurs. Deleting elements one by one and re-checking reduces W to an irreducible witness. The method rests on two assumptions: (A1) pre-existing deny statements are not attached to groups, so that removing a membership never removes a deny; (A2) legitimate holdings are judged on the original configuration, as in (8). Under A1 and A2 every rule only adds facts when elements are present, the closure is monotone, and

$$C \setminus x \text{ is safe} \Rightarrow x \cap W \neq \emptyset \quad \text{for every witness } W. \quad (9)$$

If some witness were left untouched, its derivation would still be available in $C \setminus x$, because removing other elements cannot disable a monotone derivation. Both assumptions hold in all configurations used in this paper; the IAM Vulnerable export has deny statements on six principals and none on groups.

Usage and operational impact. Usage evidence is a set P of pairs $p = (e, a, \rho)$ with counts c_p observed over a window, for example 90 days of last-accessed data or CloudTrail events. Assumption of a role by a user or by a service is also a pair. Each pair receives the weight

$$w_p = 1 + \ln(1 + c_p), \quad (10)$$

so that frequent use counts more without letting a single busy pipeline dominate. A route of p is a set of



elements that legitimately provides it: a direct grant; a membership together with the group grant; a trust entry, possibly with the identity grant for sts:AssumeRole; and for privileges of a role, additionally one of the observed routes by which the role is entered (the product is capped at 64 routes per pair, which can only overstate impact and was not reached on the real export). Let $Rt(p)$ be the set of routes. A repair breaks p when it adds a deny that covers p or hits every route:

$$b_p(x) = 1 \Leftrightarrow (\exists y \in x \cap Y: y \text{ covers } p) \vee \forall r \in Rt(p): r \cap x \neq \emptyset. \quad (11)$$

The operational impact of a repair is

$$I(x) = \sum_{p \in P} w_p b_p(x). \quad (12)$$

AWS last-accessed information does not record iam:PassRole and covers management events only [9]. For such actions the method can add a conservative prior: every held but unobserved grant of an unobservable action gets a pseudo pair with a small count (three in our experiments), which protects the grant from deletion but not from a narrow deny on a resource that was never used.

Problem P1, usage-aware safe repair. Given the set $\mathcal{W}$ of witnesses,

$$x^* = \operatorname{argmin}_{x \subseteq X} I(x) + \varepsilon |x| \quad \text{s.t.} \quad x \cap W \neq \emptyset \quad \forall W \in \mathcal{W}, \quad (13)$$

where $\varepsilon > 0$ prices each change. Since the smallest pair weight is $1 + \ln 2 \approx 1.69$, with $\varepsilon = 0.01$ the solver accepts up to about 169 extra changes to avoid one broken use; larger ε moves the solution toward small patches. The objective of IAMPERE [2] is $\min |x|$ under the same constraints; we also use its lexicographic refinement, minimum size first and minimum impact second (MinSize-Lex), to separate the effect of usage from tie-breaking.

Problem P2, budgeted risk reduction. When a change window allows at most B operations,

$$\max \sum_{(u,\theta)} \omega_\theta s_{u\theta} - \mu I(x) - \nu |x| \quad \text{s.t.} \quad |x| \leq B, \quad s_{u\theta} \leq |x \cap W| \quad \forall W \in \mathcal{W}_{u\theta}, \quad (14)$$

where $s_{u\theta} \in \{0,1\}$ marks an escalation as secured and $\mathcal{W}_{u\theta}$ are its witnesses (a witness of $\text{Full}(u)$ belongs to every target of u). We set $\mu = 0.01$ and $\nu = 10^{-5}$ relative to $\omega_\theta \in \{0.6,1\}$. Risk then dominates as long as an impact difference stays below $0.6/\mu = 60$, which holds in all budgeted experiments (impacts below 10); impact breaks ties among equally secure repairs.

Linear encoding. With binary x_o for each vocabulary element, an auxiliary h_{pr} for each route and b_p for each pair,

$$h_{pr} \geq x_o \quad \forall o \in r, \quad b_p \geq \sum_{r \in Rt(p)} h_{pr} - (|Rt(p)| - 1), \quad b_p \geq x_y \quad \forall y \text{ covering } p. \quad (15)$$

Since b_p has a positive cost, the solver sets it to one exactly when (11) holds. Only elements that occur in some witness receive variables. A pair with a route that contains none of them cannot be broken by removals; it is kept only if a deny in the vocabulary covers it.

Counterexample-guided solving. The witness set is not enumerated in advance. Starting from $x = \emptyset$, the procedure (Fig. 1) computes the closure from every $u \in U_0$ in $C \setminus x$, turns each escalation into an irreducible witness, and enumerates up to eight alternative witnesses per escalation by provisionally blocking the deny elements of the previous one; this matters when a wildcard grant allows the same attack against many resources. The witnesses become clauses, CP-SAT [32] solves (13) or (14), and the loop repeats until $C \setminus x$ is safe (for P2, until every escalation claimed as secured is verified), with a limit of 200 iterations that was never reached. By (9) the final x is optimal for P1 among all repairs in the vocabulary, because the relaxation over the collected witnesses contains every safe repair. For P1 we cross-check the optimum with the RC2 MaxSAT solver of PySAT [33] on the final witness set.

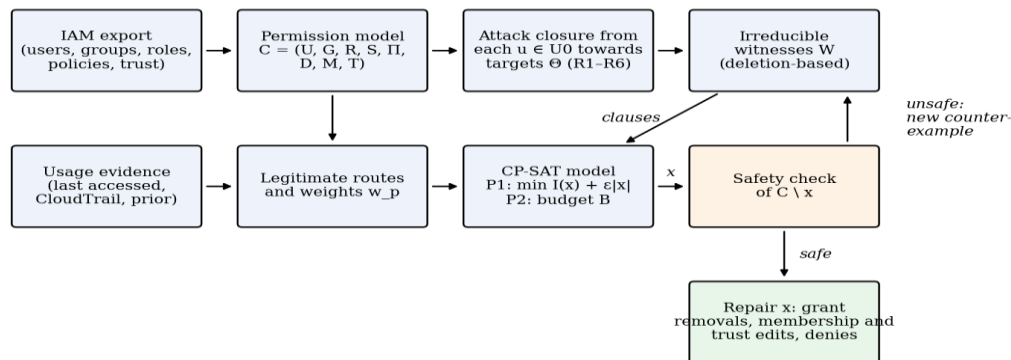


Fig. 1. Workflow of the usage-aware repair method. Inputs (left) produce the permission model and legitimate routes; the counterexample-guided loop (right) alternates between witness extraction and optimization until the repaired configuration is safe.

The procedure returns concrete edits: grants to remove, memberships or trust entries to delete, and deny statements to add, each traceable to the escalations it closes and the usage it affects.

METHODOLOGY OF THE STUDY

Research questions. RQ1: how much legitimate usage does a minimum-size repair break compared with usage-aware repair at equal safety? RQ2: how many more changes does usage-aware repair make, and how does ϵ trade changes against impact? RQ3: how much escalation risk can be removed with a small change budget? RQ4: how sensitive is the result to incomplete evidence (shorter windows, unrecorded iam:PassRole) and to the usage model, and does the conservative prior help? RQ5: how does solving time grow with account size? RQ6: does the model detect the documented escalations in a real IAM export, and how do the methods compare on it?

Synthetic organizations. No public dataset combines real IAM configurations with action-level usage, so we wrote a generator (Table 2). An organization has six teams; each team has a developer group with job privileges (team bucket, logs, function invocation), four to twelve users, an application role trusted by Lambda or EC2, and a CI role that the team's users may assume and that may pass the application role and create functions and stacks. Three administrators belong to an administrators group with full rights and may assume two administrator roles; an automation role with full rights is trusted by Lambda; a data role holds the crown-jewel privileges and is trusted by EC2. Targets are administrator equivalence ($\omega = 1$) and three sensitive data privileges ($\omega = 0.6$). Misconfigurations are drawn uniformly from eight patterns and placed on a user (30%), the team group (40%) or the CI role (30%). True usage over 90 days is generated per principal. In the default model the legitimate part of a risky grant is always a narrow, non-privileged resource (for instance, passing the team's own application role); a second model (RQ4) lets that use target a privileged resource with probability q , which no repair can then preserve without leaving a path. Seeds are fixed.

Table 2

Generator parameters and injected misconfiguration patterns

Parameter or pattern	Value or grant added	Escalation rule
Teams, users per team, administrators	6; 4 to 12; 3	
Principals per organization (50 seeds)	56 to 80, median 66	
Number of misconfigurations	$\max(1, \text{round}(N(0.06n, (0.06n)^{0.5})))$, n = non-administrator users and team roles	
Placement of a misconfiguration	user 30%, team group 40%, CI role 30%	
Use of job privileges; assumption of the CI role by a team user	0.85; 0.6	
Use of the legitimate part of a risky grant per holder	0.35 (varied 0 to 1)	
Usage counts over 90 days	$\max(1, \lfloor \lognormal(2.0, 1.2) \rfloor)$, service and automation roles +5 or +20	



Parameter or pattern	Value or grant added	Escalation rule
passrole_star	iam:PassRole on *, lambda:CreateFunction, lambda:InvokeFunction	R5
self_policy	Attach/Put policy on own user or own group; iam:PutRolePolicy on * for the CI role	R6
access_key	iam:CreateAccessKey on *	R2
add_to_admins	iam:AddUserToGroup on *	R4
trust_rewrite	iam:UpdateAssumeRolePolicy and sts:AssumeRole on *	R3
policy_version	iam:CreatePolicyVersion on *	R6
assume_star_open_trust	sts:AssumeRole on *, break-glass role trusts account root	R1
data_role_open	data role trusts account root, holder may assume it	R1

Compared methods. UA-Repair solves P1 with $\varepsilon = 0.01$. MinSize solves the IAMPERE objective $\min|x|$ with the same vocabulary, witnesses and solver, so that only the objective differs; we did not run the IAMPERE implementation itself, because its released version works on abstract tasks without usage data and cannot express deny statements. MinSize-Lex adds impact as a secondary criterion. Greedy-UA is a counterexample-guided greedy hitting set that picks the element with the best ratio of newly hit witnesses to marginal impact. Deny-on-principal follows the strategy of IAM-Deescalate [8]: for every escalation witness it adds one deny statement, on the principal that performs the escalating step (an IAM write, credential takeover or iam:PassRole action, or role assumption if that is the escalating step), scoped to the specific resource used, and repeats until no escalation remains. It is a re-implementation of the strategy on our model, not the tool. For P2, UA-Budget is compared with Size-Budget ($\mu = 0$) and with Greedy-UA and Deny-on-principal stopped after B operations.

Evidence settings (RQ4). The method sees thinned counts in a 90-, 30- or 7-day window with iam:PassRole removed, optionally with the conservative prior. Impact is always measured on the true usage. In the oracle setting (RQ1 to RQ3) the method sees the usage on which it is scored, so these results are an upper bound, and RQ4 measures how much of the benefit survives with realistic evidence.

Real export (RQ6). We used the authorization details export of an AWS account with IAM Vulnerable deployed, published in the Cloudsplaining repository [34] (44 users, 5 groups, 78 roles, 84 managed policies). The importer handles inline and managed policies, Allow and Deny statements, Action and NotAction, ARN patterns and trust policies; conditions are ignored and NotResource statements are skipped. By IAM Vulnerable naming, privesc and fn principals should escalate and fp principals should not; seven techniques (14 principals) that rely on existing resources or data-plane access are outside the model. The export has no action-level usage, so usage was synthesized (20 seeds) under the narrow-use assumption: probability 0.6 for each held non-risky privilege and 0.35 for the narrow part of a risky one. Service-linked roles are protected. The two heuristics were stopped after 45 s.

Metrics and statistics. Impact $I(x)$, broken pairs, principals with at least one broken pair, residual escalations, operations $|x|$, edited policy statements (denies and removals grouped by entity and action) and edited documents (identity policy, membership list or trust policy of an entity). Methods are compared on the same organizations with the Wilcoxon signed-rank test on per-organization impact (zero differences dropped; SciPy chooses the exact or normal-approximation variant); the matched-pairs rank-biserial correlation r is reported with a positive sign when the difference favors UA-Repair. All exact solves used one CP-SAT worker and seed 0, with a time limit of 60 to 600 s per solver call; every completed solve ended with a proof of optimality.

Environment. Python 3.10, OR-Tools 9.15 (CP-SAT) and PySAT 1.9.dev16 (RC2) on a 4-core Linux virtual machine with 3 GB of memory. For RQ5 each instance ran in its own process with a wall budget of 150 s. Eleven unit tests check the rules, the irreducibility and soundness of witnesses, the safety of all repairs and the optimality of P1 against brute force on a small case.

RESULTS OF THE STUDY

Safety and exactness. In all 50 organizations every method reached zero residual escalations, and all CP-SAT solves were proven optimal. The RC2 optimum of P1 agreed with CP-SAT to within 0.002, the rounding of integer-scaled weights. UA-Repair needed a median of 3 iterations (maximum 13) and 31.5 witnesses.

RQ1, impact. Table 3 summarizes the full repairs under oracle usage. A minimum-size repair broke on average 2.88 legitimate uses per organization and was harmless in 28% of organizations. UA-Repair broke 0.64 uses and was harmless in 76%. Over all organizations it broke 32 uses against 144 for MinSize (78% fewer; impact 79.0% lower), it was better in 32 organizations and never worse (Wilcoxon $p = 7.9 \times 10^{-7}$, $r = 1$). Breaking ties by impact recovered only 19 of the 112 avoidable broken uses: MinSize-Lex still broke 125, and UA-Repair improved on it in 29 organizations (impact 76.0% lower, $p = 3.7 \times 10^{-9}$).



The 32 uses that UA-Repair still breaks are unavoidable within the vocabulary: 26 are iam:PutGroupPolicy calls by a user on its own group, five are iam:AttachUserPolicy calls by a user on itself, and one is a CI role writing a policy to its team application role. In each case the legitimate use is itself the escalating step.

Table 3

Full repair on 50 synthetic organizations, oracle usage (impact and broken uses measured on true usage)

Method	Impact, mean	Broken uses, mean	Principals affected, mean	Zero-impact orgs	Operations, median (mean)	Edited statements, median	Time, median, s
MinSize	9.44	2.88	2.80	28%	3 (3.4)	3	0.02
MinSize-Lex	8.25	2.50	2.46	32%	3 (3.4)	3	0.02
UA-Repair	1.98	0.64	0.64	76%	8 (11.6)	6.5	0.03
Greedy-UA	1.98	0.64	0.64	76%	14 (73.6)	8	0.03
Deny-on-principal	2.07	0.66	0.66	74%	24 (76.3)	11	0.04

Organization 0 shows the mechanism. Its team 3 group held iam:CreateAccessKey on all users, and three developers used it to rotate keys of the team's service user. MinSize removed the group grant (one change, three developers broken). UA-Repair kept the grant, denied key creation on the three administrator users for the active developers and added a wildcard deny for the inactive members. Across the whole account it used ten statements instead of four and broke only one unavoidable self-policy use.

Both heuristics that work with narrow denies came close to UA-Repair in impact: Greedy-UA matched it in every organization, and Deny-on-principal broke 33 uses against 32. They needed more edits: Deny-on-principal a median of 24 operations and 11 statements, against 8 and 6.5 for UA-Repair, and Greedy-UA 14 operations (mean 73.6), because it keeps adding narrow denies when a wildcard grant permits many targets. Relative to these heuristics, exactness buys shorter repairs rather than lower impact.

RQ2, changes and the role of ε . UA-Repair used a median of 8 operations and 6.5 edited statements, against 3 and 3 for MinSize (Table 3); 92.8% of its operations were deny statements. The weight ε was then varied on 30 organizations. With $\varepsilon \leq 0.1$ the mean impact was 1.21 at 7.5 statements. Raising ε to 0.3 saved 0.6 statements per organization and added 0.16 to the mean impact; at $\varepsilon = 1$ the impact rose to 5.17 at 3.6 statements, and $\varepsilon = 10$ coincided with MinSize-Lex (6.60 at 3.2 statements).

RQ3, change budget. Fig. 2 shows the share of escalation risk removed after B operations on 30 organizations. UA-Budget removed 68.6% with one operation, 91.6% with three, 96.7% with four and all risk with eight. Size-Budget removed exactly the same share in every organization and budget, but its impact was higher: 6.19 against 5.09 for UA-Budget at $B = 4$ ($p = 0.008$) and 7.37 against 5.46 at $B = 8$ ($p = 0.0005$). Greedy-UA, stopped at the budget, removed 50.0% at $B = 4$ and 82.1% at $B = 24$, Deny-on-principal 39.7% and 68.0%; both kept impact below 1.3, because their few operations are narrow denies. Small budgets force broad cuts: with $B = 4$ an operator removes 96.7% of the risk at a mean impact of 5.09, while the unconstrained UA-Repair on the same organizations reaches zero risk at an impact of 1.21 with about ten operations.

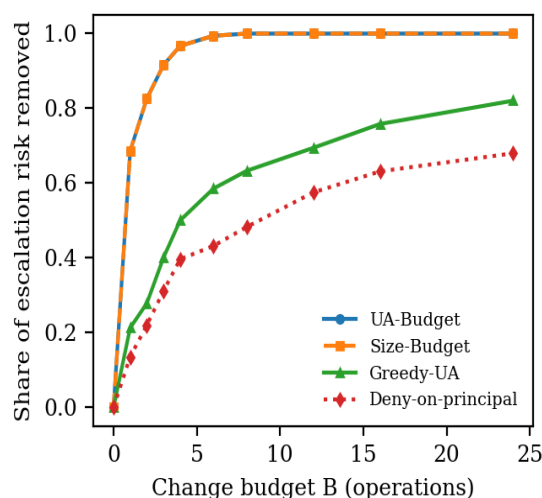


Fig. 2. Share of escalation risk removed as a function of the change budget B (30 organizations). UA-Budget and Size-Budget coincide in risk and differ only in impact.



RQ4, evidence quality and usage model. Table 4 reports impact on true usage when the method sees partial evidence. Hiding iam:PassRole alone raised the impact of UA-Repair from 1.21 to 2.18, because it then deletes CI grants that are in fact used. Shorter windows hurt more: 3.14 at 30 days and 5.31 at 7 days, where only 47% of the pairs remain visible. The conservative prior restored the oracle value at 90 days (1.21) and helped little at 7 days (4.82), since the missing pairs there are mostly observable actions that were not seen. In all settings UA-Repair stayed below MinSize (7.56), by 84.0% at 90 days with the prior and by 29.8% at 7 days without it ($p < 0.01$ everywhere).

Table 4

Effect of usage evidence on operational impact (30 organizations; mean impact on true usage)

Evidence available to the method	Visible pairs, mean	UA-Repair	MinSize-Lex	Reduction vs MinSize (7.56)
All usage (oracle)	295.9	1.21	6.60	84.0%
90 days, iam:PassRole hidden	290.4	2.18	6.86	71.2%
30 days, iam:PassRole hidden	237.0	3.14	6.86	58.5%
7 days, iam:PassRole hidden	140.1	5.31	6.93	29.8%
90 days, hidden, conservative prior	297.1	1.21	6.60	84.0%
7 days, hidden, conservative prior	146.8	4.82	6.90	36.3%

The usage model matters as much as the evidence. When the legitimate part of a risky grant sometimes targets a privileged resource (probability q , oracle evidence), the impact of UA-Repair rose from 1.21 ($q = 0$) to 2.09 ($q = 0.25$) and 2.55 ($q = 0.5$), and its advantage over MinSize shrank from 84.0% to 70.6% and 64.8%. Deny-on-principal suffered more, with impact 1.36, 2.96 and 4.25, because a deny on the escalating step cannot avoid the privileged resource, while UA-Repair may cut the path elsewhere; at $q = 0.5$ UA-Repair was 39.9% lower ($p = 0.016$).

RQ5, scalability. Up to 253 principals UA-Repair needed a median of 0.60 s, at about 485 principals 7.2 s and at about 970 principals 41.4 s for the three of five instances that finished; three larger instances exceeded 150 s, while MinSize finished all of them (median 6.1 s at 96 teams). In a profiled 48-team run, 89% of the time went to closure computation.

RQ6, IAM Vulnerable export. Of 66 labeled principals within scope, the closure detected 50 of 58 that should escalate and cleared 6 of 8 that should not. The two false positives (the fp5 pair) depend on a condition key, which the model ignores. All eight misses need a capability the principal does not have on its own, for example a role variant of a user-only technique. The closure also flagged five unlabeled principals, among them a demonstration user with iam:PassRole and lambda:* and an EC2 role allowed to create access keys for any user; both are genuine paths.

Table 5 compares the repairs on this account. UA-Repair broke 3.70 uses on average against 10.40 for MinSize and 6.55 for MinSize-Lex, an impact reduction of 65.7% and 44.8% (lower in 20 of 20 and 19 of 20 usage seeds, $p < 10^{-5}$). It used 54.5 operations and 50.5 statements against 39, and edited 35 documents against 33, in a median of 1.3 s. Neither heuristic reached a safe configuration within 45 s in any of the 20 runs: Greedy-UA left a median of 3 escalations after 185 operations, and Deny-on-principal left 4 after 1,367 denies, because wildcard grants such as iam:CreateAccessKey or iam:PassRole on all resources produce one escalation per possible target.

Table 5

Repair of the IAM Vulnerable export (104 candidate principals, 6,618 elements; 20 synthetic usage seeds, median 3,911 usage pairs)

Method	Impact, mean	Broken uses, mean	Operations, median	Statements, median	Documents edited, median	Unsafe runs	Time, median, s
MinSize	29.51	10.40	39	39	33	0 of 20	0.51
MinSize-Lex	18.31	6.55	39	39	33	0 of 20	0.51
UA-Repair	10.11	3.70	54.5	50.5	35	0 of 20	1.30
Greedy-UA (45 s)	n/a (unsafe)	n/a	185	65.5	39	20 of 20	49.1
Deny-on-principal (45 s)	n/a (unsafe)	n/a	1,367	94	57	20 of 20	45.8



Discussion. Minimizing the number of changes is a poor proxy for minimizing disruption: under oracle usage the minimum-size repair was harmless in fewer than a third of the organizations, and tie-breaking by impact recovered only a small part of the difference, because the cheapest cut is often the grant itself while the harmless cut is a narrow deny on the dangerous targets. The oracle gain is an upper bound; with realistic evidence the reduction relative to MinSize was 71.2% at 90 days and 29.8% at 7 days, against 84.0% with oracle usage.

The price is more edits, about twice as many statements as MinSize, and the trade-off is tunable through ϵ . Against heuristics that also use narrow denies the benefit is of a different kind. Under the default usage model they break about as little, but they need more edits (Deny-on-principal 11 statements against 6.5), remove much less risk per change under a budget, and on the real export did not converge within their time limit. When legitimate use touches privileged resources, the freedom to cut a path at any element, not only at the escalating step, lowered impact by up to 39.9% relative to per-principal denial.

Missing PassRole data, the normal situation in AWS, raised the impact by 80%; a conservative prior closed that gap, but short observation windows cannot be fixed by priors. PassRole usage can also be partly recovered from CloudTrail, since calls such as `lambda:CreateFunction` and `ec2:RunInstances` record the passed role.

Repairs that keep a wildcard grant and add denies on named targets are safe only for the analyzed snapshot: in organization 0, an administrator added the next day would not be covered. The analysis therefore has to run again whenever principals are added, for example as a policy-as-code check in the deployment pipeline [35, 36], or denies should be written on tags or paths (`aws:ResourceTag, /admin/`), which our vocabulary does not yet include. Our engine is exact and ran in seconds on accounts of several hundred principals; accounts with thousands of principals would need incremental closure computation or learned pruning, as in IAMPERE [2].

Limitations. The rule set covers the common control-plane techniques but not data-plane ones, conditions (also on deny statements), `NotResource`, permission boundaries, SCPs or resource policies other than trust, and launch actions are recognized only when granted on all resources; IAM Vulnerable showed both a false-positive pair and techniques outside the model. Usage is synthetic in every experiment, including the real export, and the default model assumes that legitimate use of a risky grant targets a narrow, non-privileged resource; the q -model and the evidence experiments bound, but do not remove, this threat. Repairs are at action level on a flattened model, while real edits change policy documents; deny statements need care around AWS managed policies and policy size quotas. Monotonicity assumptions A1 and A2 must be checked for each account.

CONCLUSIONS AND PROSPECTS FOR FURTHER RESEARCH

The paper proposed a usage-aware method for repairing privilege escalation paths in AWS IAM. The method models an account as a permission graph with monotone escalation rules, turns each escalation into an irreducible witness, and chooses edits that hit every witness while breaking as little observed legitimate usage as possible. Under two stated assumptions the counterexample-guided solution is exact within the repair vocabulary, and a budgeted variant gives the best risk reduction for a fixed number of changes.

In 50 synthetic organizations the method closed all escalation paths while breaking 78% fewer legitimate uses than a minimum-size repair and 74% fewer than a minimum-size repair with impact tie-breaking, at the cost of about twice as many edited statements; a per-principal deny strategy broke about as little but needed about 1.7 times as many statements. With four changes the budgeted variant removed 96.7% of escalation risk. Without PassRole evidence the impact rose from 1.21 to 2.18, and a conservative prior removed that loss. On the IAM Vulnerable export the model found 50 of 58 in-scope escalations, and usage-aware repair reduced impact by 65.7% relative to a minimum-size repair.

Further work will validate the impact model on real usage from production accounts, add tag- and path-based deny statements so that repairs stay safe as accounts change, extend the rules to conditions, permission boundaries and service control policies, recover PassRole usage from CloudTrail, and emit repairs as reviewable infrastructure-as-code changes, as is already done for firewall rule sets [36].

DECLARATION ON THE USE OF ARTIFICIAL INTELLIGENCE

During the preparation of this work the authors used Claude (Anthropic) to assist with computations in the experimental evaluation and with language editing and formatting of the manuscript. The authors checked all computed results against the source data and scripts, reviewed and edited the text, and take full responsibility for the content of the publication.



DATA AND CODE AVAILABILITY

The IAM Vulnerable export used in RQ6 is publicly available in the Cloudsplaining repository [34] (BSD-3-Clause license). The generator, model and experiment scripts are available from the corresponding author on reasonable request.

REFERENCES

1. Novaes Neto, N., Madnick, S. E., Moraes G. de Paula, A., & Malara Borges, N. (2020). *A case study of the Capital One data breach* [Working paper]. SSRN. <https://doi.org/10.2139/ssrn.3542567>
2. Hu, Y., Wang, W., Khurshid, S., McMillan, K. L., & Tiwari, M. (2023). Fixing privilege escalations in cloud access control with MaxSAT and graph neural networks. In *Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (pp. 104–115). IEEE. <https://doi.org/10.1109/ASE56229.2023.00167>
3. Gietzen, S. (2018). *AWS IAM privilege escalation: Methods and mitigation*. Rhino Security Labs. <https://rhinosecuritylabs.com/aws/aws-privilege-escalation-methods-mitigation/>
4. Datadog Security Labs. (n.d.). *pathfinding.cloud: AWS IAM privilege escalation library*. <https://pathfinding.cloud/>
5. NCC Group. (n.d.). *Principal Mapper (PMapper)* [Computer software]. GitHub. <https://github.com/nccgroup/PMapper>
6. Shevrin, I., & Margalit, O. (2023). Detecting multi-step IAM attacks in AWS environments via model checking. In *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association. <https://www.usenix.org/conference/usenixsecurity23/presentation/shevrin>
7. Hu, Y., Wang, W., Khurshid, S., & Tiwari, M. (2023). *TAC: Hybrid IAM privilege escalation detection* (arXiv:2304.14540, version 9, 2026) [Preprint]. arXiv. <https://arxiv.org/abs/2304.14540>
8. Chen, J. (2022, July 25). *IAM-Deescalate: An open source tool to help users reduce the risk of privilege escalation*. Palo Alto Networks Unit 42. <https://unit42.paloaltonetworks.com/iam-deescalate/>
9. Amazon Web Services. (n.d.). *Refine permissions in AWS using last accessed information*. AWS Identity and Access Management User Guide. https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_last-accessed.html
10. Art, S. (2021, September 9). *IAM Vulnerable: An AWS IAM privilege escalation playground*. Bishop Fox. <https://bishopfox.com/blog/aws-iam-privilege-escalation-playground>
11. Liu, X., Yang, N., Li, C., Li, J., Ma, J., & Lu, K. (2026). The dark side of flexibility: Detecting risky permission chaining attacks in serverless applications. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. Internet Society. <https://doi.org/10.14722/ndss.2026.230819>
12. Rudin, D., Vasylenko, V., & Grynkevych, G. (2025). AI agents for automated network incident detection and misconfiguration identification. *International Scientific Journal "Internauka"*, (12). <https://doi.org/10.25313/2520-2057-2025-12>
13. Backes, J., Bolognani, P., Cook, B., Dodge, C., Gacek, A., Luckow, K., Rungta, N., Tkachuk, O., & Varming, C. (2018). Semantic-based automated reasoning for AWS access policies using SMT. In *2018 Formal Methods in Computer Aided Design (FMCAD)* (pp. 1–9). IEEE. <https://doi.org/10.23919/FMCAD.2018.8602994>
14. Bouchet, M., Cook, B., Cutler, B., Druzkina, A., Gacek, A., Hadarean, L., Jhala, R., Marshall, B., Peebles, D., Rungta, N., Schlesinger, C., Stephens, C., Varming, C., & Warfield, A. (2020). Block public access: Trust safety verification of access control policies. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (pp. 281–291). ACM. <https://doi.org/10.1145/3368089.3409728>
15. D'Antoni, L., Ding, S., Goel, A., Ramesh, M., Rungta, N., & Sung, C. (2024). Automatically reducing privilege for access control policies. *Proceedings of the ACM on Programming Languages*, 8(OOPSLA2), 763–790. <https://doi.org/10.1145/3689738>
16. Eiers, W., Sankaran, G., & Bultan, T. (2023). Quantitative policy repair for access control on the cloud. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)* (pp. 564–575). ACM. <https://doi.org/10.1145/3597926.3598078>
17. Wu, K. L., Jenkins, C., Stoller, S. D., & Chowdhury, O. (2026). Automatically tightening access control policies with Restrictor. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2026)* (Lecture Notes in Computer Science, pp. 110–129). Springer. https://doi.org/10.1007/978-3-032-22752-2_6



18. Hall, B., Ungaro, O., & Eiers, W. (2025). *CloudFix: Automated policy repair for cloud access control policies using large language models* (arXiv:2512.09957) [Preprint]. arXiv. <https://arxiv.org/abs/2512.09957>
19. Sanders, M. W., & Yue, C. (2017). Automated least privileges in cloud-based web services. In *Proceedings of the Fifth ACM/IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)* (pp. 1–6). ACM. <https://doi.org/10.1145/3132465.3132470>
20. Sanders, M. W., & Yue, C. (2018). Minimizing privilege assignment errors in cloud services. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy (CODASPY)* (pp. 2–12). ACM. <https://doi.org/10.1145/3176258.3176307>
21. Vaidya, J., Atluri, V., Guo, Q., & Adam, N. (2008). Migrating to optimal RBAC with minimal perturbation. In *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies (SACMAT)* (pp. 11–20). ACM. <https://doi.org/10.1145/1377836.1377839>
22. Sheyner, O., Haines, J., Jha, S., Lippmann, R., & Wing, J. M. (2002). Automated generation and analysis of attack graphs. In *Proceedings 2002 IEEE Symposium on Security and Privacy* (pp. 273–284). IEEE. <https://doi.org/10.1109/SECPRI.2002.1004377>
23. Noel, S., Jajodia, S., O’Berry, B., & Jacobs, M. (2003). Efficient minimum-cost network hardening via exploit dependency graphs. In *19th Annual Computer Security Applications Conference (ACSAC 2003)* (pp. 86–95). IEEE. <https://doi.org/10.1109/CSAC.2003.1254313>
24. Wang, L., Noel, S., & Jajodia, S. (2006). Minimum-cost network hardening using attack graphs. *Computer Communications*, 29(18), 3812–3824. <https://doi.org/10.1016/j.comcom.2006.06.018>
25. Guo, M., Li, J., Neumann, A., Neumann, F., & Nguyen, H. (2022). Practical fixed-parameter algorithms for defending Active Directory style attack graphs. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(9), 9360–9367. <https://doi.org/10.1609/aaai.v36i9.21167>
26. Abdulraheem, W. K., Ibbini, E. M., Kanaker, H., Smadi, S., Abdel Karim, N., Fakhouri, H. N., Albdour, L., & Fakhouri, S. (2026). Least-privilege role-based access control improvement for cloud container security. *Computers*, 15(5), 326. <https://doi.org/10.3390/computers15050326>
27. Kobilnyk, B., & Sydorenko, V. (2026). Experimental study of an adaptive contextual access control model for increasing the resilience of critical information systems [in Ukrainian]. *Cybersecurity: Education, Science, Technique*, 2(34), 702–718. <https://doi.org/10.28925/2663-4023.2026.34.1364>
28. Syrotynskyi, R., & Tyshyk, I. (2026). Improving compliance with zero trust principles in enterprise network architecture using artificial intelligence [in Ukrainian]. *Cybersecurity: Education, Science, Technique*, 2(34), 630–645. <https://doi.org/10.28925/2663-4023.2026.34.1356>
29. Grynkevych, G., Vasylenko, V., & Rudin, D. (2026). Local open-source platform for cyber incident monitoring and response with AI support as an alternative to EDR/XDR solutions. *Cybersecurity: Education, Science, Technique*, 1(33), 234–241. <https://doi.org/10.28925/2663-4023.2026.33.1211>
30. Pliekhov, O., Grynkevych, G., & Vasylenko, V. (2026). An extended STRIDE threat model for offline BLE mesh disaster communication applications. *Cybersecurity: Education, Science, Technique*, 2(34), 726–743. <https://doi.org/10.28925/2663-4023.2026.34.1368>
31. Jenkyn, J., & Wadman, L. (2022, November 3). *How to use trust policies with IAM roles*. AWS Security Blog. <https://aws.amazon.com/blogs/security/how-to-use-trust-policies-with-iam-roles/>
32. Perron, L., & Furnon, V. (n.d.). *OR-Tools* (Version 9.15) [Computer software]. Google. <https://developers.google.com/optimization>
33. Ignatiev, A., Morgado, A., & Marques-Silva, J. (2018). PySAT: A Python toolkit for prototyping with SAT oracles. In *Theory and Applications of Satisfiability Testing (SAT 2018)* (Lecture Notes in Computer Science, Vol. 10929, pp. 428–437). Springer. https://doi.org/10.1007/978-3-319-94144-8_26
34. Salesforce. (n.d.). *Cloudsplaining: AWS IAM security assessment tool* [Computer software]. GitHub. <https://github.com/salesforce/cloudsplaining>
35. Rudin, D., Vasylenko, V., & Grynkevych, G. (2026). Adaptive network security automation: DevSecOps, ML detection, and policy-as-code. *International Scientific Journal “Internauka”*, (1). <https://doi.org/10.25313/2520-2057-2026-1>
36. Grynkevych, G., Vasylenko, V., Rudin, D., & Pliekhov, O. (2026). Design and implementation of automated firewall policy management in enterprise telecommunication systems using infrastructure-as-code. *Zviazok*, (4), 3–12. <https://doi.org/10.31673/2412-9070.2026.049412>

**Гринкевич Ганна Олександрівна**

доктор технічних наук, доцент,
професор кафедри інформаційної та кібернетичної безпеки імені професора Володимира Бурячка
Київський столичний університет імені Бориса Грінченка, Київ, Україна
ORCID: 0000-0003-1922-5165
ggrynkevyich@ukr.net

Василенко Володимир Вікторович

кандидат технічних наук, доцент,
доцент кафедри комп'ютерних наук
Державний університет інформаційно-комунікаційних технологій, Київ, Україна
ORCID: 0000-0001-8465-6178
oknelisavvova172@gmail.com

**МЕТОД УСУНЕННЯ ШЛЯХІВ ПІДВИЩЕННЯ ПРИВІЛЕЇВ У СИСТЕМІ КЕРУВАННЯ
ІДЕНТИФІКАЦІЄЮ ТА ДОСТУПОМ AWS З УРАХУВАННЯМ ФАКТИЧНОГО
ВИКОРИСТАННЯ ДОЗВОЛІВ**

Анотація. Помилки в налаштуванні політик ідентифікації Amazon Web Services (AWS) дають змогу користувачеві або ролі з низькими привілеями отримати права адміністратора через ланцюжки дозволів, кожен з яких окремо виглядає безпечним, наприклад iam:PassRole чи iam:CreateAccessKey. Наявні методи обирають виправлення за кількістю змін у політиках, тому «мінімальне» виправлення може порушити роботу конвеєрів розгортання, сервісних ролей і людей, які користуються дозволом легітимно. У статті запропоновано метод усунення шляхів підвищення привілеїв у системі керування ідентифікацією та доступом (IAM) AWS з урахуванням фактичного використання дозволів. Обліковий запис подано у вигляді графа дозволів із монотонними правилами ескалації, а дані журналів доступу визначають операційний вплив як зважену кількість спостережених використань (принципал, дія, ресурс), які порушить виправлення. Задачу розв'язано як точну оптимізацію CP-SAT, керовану контрприкладом: модель або мінімізує вплив за умови повної безпеки, або максимізує усунення ризик за обмеженої кількості змін. За сформульованих припущень монотонності розв'язок оптимальний у межах словника виправлень. Для 50 синтетичних організацій метод усунув усі ескалації і порушив на 78% менше легітимних використань, ніж виправлення мінімального розміру (32 проти 144; критерій Вілкоксона, $p < 10^{-6}$), ціною 6,5 замість 3 змінених положень політик (медіана). Приховування даних про iam:PassRole, як це відбувається в даних AWS про останній доступ, збільшило вплив з 1,21 до 2,18, а консервативне апріорне припущення для неспостережуваних дій повернуло його до 1,21. З чотирма змінами варіант з обмеженням усунув 96,7% ризику ескалації. На відкритому експорті облікового запису IAM Vulnerable модель виявила 50 з 58 сценаріїв ескалації, які вона охоплює, а виправлення з урахуванням використання зменшило вплив на 65,7% порівняно з мінімальним.

Ключові слова: безпека хмарних обчислень; AWS IAM; підвищення привілеїв; принцип найменших привілеїв; виправлення політик доступу; MaxSAT; програмування з обмеженнями; журнали використання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Novaes Neto, N., Madnick, S. E., Moraes G. de Paula, A., & Malara Borges, N. (2020). *A case study of the Capital One data breach* [Working paper]. SSRN. <https://doi.org/10.2139/ssrn.3542567>
2. Hu, Y., Wang, W., Khurshid, S., McMillan, K. L., & Tiwari, M. (2023). Fixing privilege escalations in cloud access control with MaxSAT and graph neural networks. In *Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (pp. 104–115). IEEE. <https://doi.org/10.1109/ASE56229.2023.00167>
3. Gietzen, S. (2018). *AWS IAM privilege escalation: Methods and mitigation*. Rhino Security Labs. <https://rhinosecuritylabs.com/aws/aws-privilege-escalation-methods-mitigation/>
4. Datadog Security Labs. (n.d.). *pathfinding.cloud: AWS IAM privilege escalation library*.



- <https://pathfinding.cloud/>
5. NCC Group. (n.d.). *Principal Mapper (PMapper)* [Computer software]. GitHub. <https://github.com/nccgroup/PMapper>
 6. Shevrin, I., & Margalit, O. (2023). Detecting multi-step IAM attacks in AWS environments via model checking. In *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association. <https://www.usenix.org/conference/usenixsecurity23/presentation/shevrin>
 7. Hu, Y., Wang, W., Khurshid, S., & Tiwari, M. (2023). *TAC: Hybrid IAM privilege escalation detection* (arXiv:2304.14540, version 9, 2026) [Preprint]. arXiv. <https://arxiv.org/abs/2304.14540>
 8. Chen, J. (2022, July 25). *IAM-Deescalate: An open source tool to help users reduce the risk of privilege escalation*. Palo Alto Networks Unit 42. <https://unit42.paloaltonetworks.com/iam-deescalate/>
 9. Amazon Web Services. (n.d.). *Refine permissions in AWS using last accessed information*. AWS Identity and Access Management User Guide. https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_last-accessed.html
 10. Art, S. (2021, September 9). *IAM Vulnerable: An AWS IAM privilege escalation playground*. Bishop Fox. <https://bishopfox.com/blog/aws-iam-privilege-escalation-playground>
 11. Liu, X., Yang, N., Li, C., Li, J., Ma, J., & Lu, K. (2026). The dark side of flexibility: Detecting risky permission chaining attacks in serverless applications. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. Internet Society. <https://doi.org/10.14722/ndss.2026.230819>
 12. Rudin, D., Vasylenko, V., & Grynkevych, G. (2025). AI agents for automated network incident detection and misconfiguration identification. *International Scientific Journal "Internauka"*, (12). <https://doi.org/10.25313/2520-2057-2025-12>
 13. Backes, J., Bolignano, P., Cook, B., Dodge, C., Gacek, A., Luckow, K., Rungta, N., Tkachuk, O., & Varming, C. (2018). Semantic-based automated reasoning for AWS access policies using SMT. In *2018 Formal Methods in Computer Aided Design (FMCAD)* (pp. 1–9). IEEE. <https://doi.org/10.23919/FMCAD.2018.8602994>
 14. Bouchet, M., Cook, B., Cutler, B., Druzkina, A., Gacek, A., Hadarean, L., Jhala, R., Marshall, B., Peebles, D., Rungta, N., Schlesinger, C., Stephens, C., Varming, C., & Warfield, A. (2020). Block public access: Trust safety verification of access control policies. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (pp. 281–291). ACM. <https://doi.org/10.1145/3368089.3409728>
 15. D'Antoni, L., Ding, S., Goel, A., Ramesh, M., Rungta, N., & Sung, C. (2024). Automatically reducing privilege for access control policies. *Proceedings of the ACM on Programming Languages*, 8(OOPSLA2), 763–790. <https://doi.org/10.1145/3689738>
 16. Eiers, W., Sankaran, G., & Bultan, T. (2023). Quantitative policy repair for access control on the cloud. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)* (pp. 564–575). ACM. <https://doi.org/10.1145/3597926.3598078>
 17. Wu, K. L., Jenkins, C., Stoller, S. D., & Chowdhury, O. (2026). Automatically tightening access control policies with Restrictor. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2026)* (Lecture Notes in Computer Science, pp. 110–129). Springer. https://doi.org/10.1007/978-3-032-22752-2_6
 18. Hall, B., Ungaro, O., & Eiers, W. (2025). *CloudFix: Automated policy repair for cloud access control policies using large language models* (arXiv:2512.09957) [Preprint]. arXiv. <https://arxiv.org/abs/2512.09957>
 19. Sanders, M. W., & Yue, C. (2017). Automated least privileges in cloud-based web services. In *Proceedings of the Fifth ACM/IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)* (pp. 1–6). ACM. <https://doi.org/10.1145/3132465.3132470>
 20. Sanders, M. W., & Yue, C. (2018). Minimizing privilege assignment errors in cloud services. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy (CODASPY)* (pp. 2–12). ACM. <https://doi.org/10.1145/3176258.3176307>
 21. Vaidya, J., Atluri, V., Guo, Q., & Adam, N. (2008). Migrating to optimal RBAC with minimal perturbation. In *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies (SACMAT)* (pp. 11–20). ACM. <https://doi.org/10.1145/1377836.1377839>
 22. Sheyner, O., Haines, J., Jha, S., Lippmann, R., & Wing, J. M. (2002). Automated generation and analysis of attack graphs. In *Proceedings 2002 IEEE Symposium on Security and Privacy* (pp. 273–284). IEEE. <https://doi.org/10.1109/SECPRI.2002.1004377>
 23. Noel, S., Jajodia, S., O'Berry, B., & Jacobs, M. (2003). Efficient minimum-cost network hardening via exploit dependency graphs. In *19th Annual Computer Security Applications Conference (ACSAC 2003)* (pp. 86–95). IEEE. <https://doi.org/10.1109/CSAC.2003.1254313>



24. Wang, L., Noel, S., & Jajodia, S. (2006). Minimum-cost network hardening using attack graphs. *Computer Communications*, 29(18), 3812–3824. <https://doi.org/10.1016/j.comcom.2006.06.018>
25. Guo, M., Li, J., Neumann, A., Neumann, F., & Nguyen, H. (2022). Practical fixed-parameter algorithms for defending Active Directory style attack graphs. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(9), 9360–9367. <https://doi.org/10.1609/aaai.v36i9.21167>
26. Abdulraheem, W. K., Ibbini, E. M., Kanaker, H., Smadi, S., Abdel Karim, N., Fakhouri, H. N., Albdour, L., & Fakhouri, S. (2026). Least-privilege role-based access control improvement for cloud container security. *Computers*, 15(5), 326. <https://doi.org/10.3390/computers15050326>
27. Kobilnyk, B., & Sydorenko, V. (2026). Experimental study of an adaptive contextual access control model for increasing the resilience of critical information systems [in Ukrainian]. *Cybersecurity: Education, Science, Technique*, 2(34), 702–718. <https://doi.org/10.28925/2663-4023.2026.34.1364>
28. Syrotynskyi, R., & Tyshyk, I. (2026). Improving compliance with zero trust principles in enterprise network architecture using artificial intelligence [in Ukrainian]. *Cybersecurity: Education, Science, Technique*, 2(34), 630–645. <https://doi.org/10.28925/2663-4023.2026.34.1356>
29. Grynkevych, G., Vasylenko, V., & Rudin, D. (2026). Local open-source platform for cyber incident monitoring and response with AI support as an alternative to EDR/XDR solutions. *Cybersecurity: Education, Science, Technique*, 1(33), 234–241. <https://doi.org/10.28925/2663-4023.2026.33.1211>
30. Pliekhov, O., Grynkevych, G., & Vasylenko, V. (2026). An extended STRIDE threat model for offline BLE mesh disaster communication applications. *Cybersecurity: Education, Science, Technique*, 2(34), 726–743. <https://doi.org/10.28925/2663-4023.2026.34.1368>
31. Jenkyn, J., & Wadman, L. (2022, November 3). *How to use trust policies with IAM roles*. AWS Security Blog. <https://aws.amazon.com/blogs/security/how-to-use-trust-policies-with-iam-roles/>
32. Perron, L., & Furnon, V. (n.d.). *OR-Tools* (Version 9.15) [Computer software]. Google. <https://developers.google.com/optimization>
33. Ignatiev, A., Morgado, A., & Marques-Silva, J. (2018). PySAT: A Python toolkit for prototyping with SAT oracles. In *Theory and Applications of Satisfiability Testing (SAT 2018)* (Lecture Notes in Computer Science, Vol. 10929, pp. 428–437). Springer. https://doi.org/10.1007/978-3-319-94144-8_26
34. Salesforce. (n.d.). *Cloudsplaining: AWS IAM security assessment tool* [Computer software]. GitHub. <https://github.com/salesforce/cloudsplaining>
35. Rudin, D., Vasylenko, V., & Grynkevych, G. (2026). Adaptive network security automation: DevSecOps, ML detection, and policy-as-code. *International Scientific Journal "Internauka"*, (1). <https://doi.org/10.25313/2520-2057-2026-1>
36. Grynkevych, G., Vasylenko, V., Rudin, D., & Pliekhov, O. (2026). Design and implementation of automated firewall policy management in enterprise telecommunication systems using infrastructure-as-code. *Zviazok*, (4), 3–12. <https://doi.org/10.31673/2412-9070.2026.049412>

Отримано редакцією журналу / Received: 03.09.26

Прорецензовано / Revised: 16.09.26

Схвалено до друку / Accepted: 26.10.26



This work is licensed under Creative Commons Attribution-noncommercial-sharealike 4.0 International License.