# One Approach to Risk Management of Privileged Access for Linux-based Operating Systems

DMITRY A. BELYAVSKIY, ARTEM S. KABANOV, ALEXEY B. LOS
Moscow Institute of Electronics and Mathematics (MIEM),
HSE University (National Research University Higher School of Economics),
34 Tallinskaya St., Moscow, 123458,
RUSSIA

*Abstract:* - Privileged access is the standard practice of granting users access to infrastructure components, such as information system servers, in order to make changes to their configurations. From an information security perspective, this privileged access creates potential threats and the need for additional control. This article proposes a methodology for identifying privileges when users access Linux-based operating systems that require additional control. This methodology is based on a metric that takes into account the specifics of commands users run and allows calculating the levels of potential risk for each user. This approach allows the calculated estimates to be used in the information security risk management processes in a particular organization.

*Key-Words:* - privileged access, information security, threat, information security risk, privileged access management, operating system, risk assessment.

Received: February 26, 2026. Revised: May 6, 2026. Accepted: July 16, 2026. Published: November 24, 2026.

## 1 Introduction

A modern approach to information security management is an approach based on information security risk management and considering the probability of threats being realized, as well as the possibility of exploiting relevant vulnerabilities in information systems.

The main paradigm in the development of information systems in most cases is the "client-server" pair, where Linux-based operating systems are used as a "server", both public distributions (Ubuntu, Debian, Fedora) and proprietary (Red Hat, Oracle).

In turn, access to each operating system from the information systems is created for privileged users, including administrators of information systems, who perform actions to configure and maintain the corresponding operating systems. Numerous studies are presented, the essence of which is that the main source of threats to information security is internal intruders [1], and, especially, users with extended access rights – privileged users.

The use of risk-oriented models [2] for access management, in contrast to role models, makes it possible to respond to threats in almost real time [3].

*Related work.* Existing approaches to the problem fall into two groups. User and entity behavior analytics (UEBA) [1], [4], [5] learns a statistical or topological baseline of user activity and reports deviations from it; such scores are data-driven, but any deviation is treated as suspicious, so that rare yet legitimate administrative commands generate false positives, and the score carries no information about the potential damage of the action. Risk-based access control [2], [3], [6], [7] estimates risk from the properties of the request and of the environment, but the existing models operate at the level of access requests or resources rather than at the level of individual operating-system commands and their parameters. The approach proposed here lies between the two: it is command-level and damage-aware, as risk-based access control is, and it uses empirical command frequencies, as UEBA does, but as a weight applied to the damage component rather than as the sole anomaly criterion.

*Contribution.* The contribution of the paper is a command-level risk metric for privileged sessions with (i) a formally defined and bounded scoring function, (ii) proven monotonicity, order-preservation, and stability properties, (iii) an explicit treatment of the frequency component that separates server-local from infrastructure-wide baselines, and (iv) a worked example and an evaluation on six months of command traces from seven production servers (Section 4). We position the work as a risk metric with empirical validation rather than as a probabilistic model of attacker behavior. The

limitations of this positioning are discussed in Section 5.

The methodology presented in the paper proposes to combine the role-based access model with the risk assessment of user actions, comparing each executed command (action) with additional parameters in order to identify potentially "malicious" actions, that is, actions that threaten the functioning of information systems [6].

## 2 Analysis of Command Syntax

The first step of the methodology is to analyze users' actions (commands) in a Linux-based operating system [8], in which each performed user action is examined according to the scheme (example):

1. Completion of commands:
   - multirow command ("\" or output direction change with "<< EOF … EOF" (or similar));
   - non-paired (unclosed) quotation marks.
2. Splitting of combined (complex) commands with logical operators:
   - || – logical OR;
   - && – logical AND;
   - ; - separation of commands;
   - | - passing output of one command to another.
3. Replacement of command aliases and shell functions by their definitions. Aliases are not part of the command semantics: they are defined per operating system distribution, and can also be customized per user and/or per session. For example, the default ~/.bashrc shipped with Ubuntu defines:
   - "ll" equals "ls -alF";
   - "la" equals "ls -A";

   whereas a user-defined alias such as "update" equals "apt update && apt upgrade -y" exists only in the environment of the user. Consequently, alias definitions are not assumed but collected at the point of observation: (i) when commands are recorded at the level of the exec system call (see Section 2.1), aliases and functions have already been expanded by the shell, and the observed record is the actual program with its arguments; (ii) when only shell history or a session recording is available, the effective alias table is captured at session start by running "alias" in the user context and by parsing the system-wide and user initialization files (/etc/bash.bashrc, /etc/profile.d/*, ~/.bashrc, ~/.bash_aliases), and the recorded command lines are expanded against this table.
4. Analysis of the command's structure and parameters, since such parameters can significantly change the purpose of the command and impact on the system:
   - -r or -R – for recursion (repeating command for folders inside);
   - "stop" or "start" – for the "systemctl" command.

The same flag may have different meanings for different commands: "-f" means "force" for rm, cp and mv, but "archive file" for tar and "foreground" for ssh; "-y" is specific to package managers such as apt or yum. Flags are therefore not interpreted by their spelling. Each (command, flag) pair is mapped, by a command-specific dictionary derived from the command documentation, to a semantic tag — force, recursive, auto-confirm, privilege escalation, help/version and so on — and the coefficients of Section 3 are attached to the tags rather than to the flag strings. Table 3 and Table 4 (Appendix) list the tags together with the flag spellings that map to them for the most common commands; a flag with no entry in the dictionary for a given command is treated as neutral (coefficient equals 1).

Commands context parameters have been provided in 0.

Table 1. Commands context parameters

| Context parameters | Example | Indicator type |
|---|---|---|
| Privilege escalation | sudo | command token |
| Accessing system directories | /etc, /usr, /var, /lib, /opt, etc. | path |
| Accessing configuration files | .conf, .cfg, .ini, .yaml, .json, .xml, .lua | file extension |
| Network communications | curl, wget, ssh, scp, sftp, nc | command token |
| Accessing temporary directories | /tmp, /var/tmp, /dev/shm | path |
| Certificates and keys manipulation | openssl, ssh-keygen; ~/.ssh; .crt, .key, .pem, id_rsa* | command token; path; file extension |
| Service management | systemctl, service | command token |
| Cron scheduler management | crontab -e | command token + parameter |
| Scripts execution | .sh, .pl, .py | file extension |
| Interpreter invocation with inline code | bash -c, sh -c, python -c, perl -e | command token + parameter |
| Encoded or obfuscated input | base64 -d, xxd -r, eval | command token |

*Source: created by the authors*

Each parameter is recorded as a binary and is used in subsequent stages of risk calculation. After performing semantic analysis, each atomic operation receives:

- Base command;

- Normalized structure of command;
- Parameters;
- Set of context parameters.

These data form the basis for the subsequent calculation of the integral risk value.

Commands could be analyzed during an active user session (in real time) or after the sessions end.

### 2.1 Observation Point, Normalization and Adversarial Evasion

The command representation described above is the foundation of the risk model; we therefore state explicitly what the normalization procedure does and does not guarantee. The overall processing pipeline, from raw records to the risk indicators of Section 3, is shown in Figure 2 (Appendix).

*Observation point.* Two sources of command records are considered. The primary source is the audit subsystem of the kernel (Linux Audit [9] or an equivalent eBPF-based tracer) configured to record every exec system call together with its arguments, the resolved path of the executable, and the identity of the calling user and session. The secondary source is shell history or a session recording, which is what many PAM products provide. The difference between the two is essential for evasion resistance. At the exec level, the shell has already performed alias and function expansion, quote removal, parameter and variable expansion, command substitution, globbing, and the splitting of pipelines, lists and subshells into separate processes, so that the observed record is the actual program together with its actual arguments. None of these constructs can therefore be used to disguise a command from an exec-level observer. The lexical procedure of this section is needed only for the secondary source, where it implements the corresponding rules of the POSIX shell grammar [10] — quoting, the operators "|", "||", "&&" and ";", redirections, here-documents and line continuation — on a best-effort basis.

*What remains after expansion.* The following constructs are not resolved by the shell and are handled by the normalization procedure as far as possible:

- interpreter invocation with inline code (bash -c, sh -c, python -c, perl -e): the code argument is extracted and parsed recursively as a command line; if it cannot be parsed, the record keeps the context indicator "interpreter invocation" (Table 1);
- scripts (bash script.sh, ./script.sh, . script.sh): the invocation receives the indicator "script execution"; when the script body is available on the host, it is analyzed as a sequence of commands and the maximum risk over its commands is attributed to the invocation;
- encoded or obfuscated input (base64 -d, xxd -r, eval, printf with escape sequences piped into a shell): the record receives the indicator "encoded input", which is treated as a risk-increasing parameter;
- indirect execution through symbolic links, copied or renamed binaries: the audit record contains the resolved path of the executable, which is used in place of the typed name;
- shell built-ins (cd, export, ulimit, umask) produce no exec record; they are taken from shell history when available and are of low risk in the model in any case.

*Limitations.* We do not claim that the normalization preserves the security-relevant semantics of arbitrary shell input. Shell semantics depend on the environment (variables, current directory, file system state) and on the interpreted code that a command may load, and a formal proof of semantic equivalence between the recorded and the normalized command is out of the scope of this paper. The procedure is a lexical normalizer designed for the common case. Obfuscation inside interpreted code, environment-dependent behavior, and binaries replaced by a user who already holds root privileges are treated as residual evasions and addressed by two mechanisms of the model rather than by the parser: the rarity weight (19), which assigns the maximum weight to commands never observed on the server, and the context parameters of Table 1, which flag the evasion techniques themselves as risk-increasing parameters. A user who knows the scoring scheme can thus lower the structural score Q of a command only by using commands that are themselves scored and that, being unusual, are not discounted by the frequency weight. A systematic evaluation of evasion resistance is left for future work.

## 3 Methodology

Each command belongs to a specific command family. For example:

- data deletion;
- service management;
- network communications;
- configuration changes;
- routine operations.

For each command family in this methodology, a base risk value is introduced – a numerical value

that reflects a typical potential assessment of the undesirability of actions of this class, that is, an assessment of the direction of the user's actions in violating the confidentiality, integrity or availability of information processed on the server (or servers).

D. Dubois in the computer theory of probability [11] uses the term "uncertainty" in relation to fuzzy logic and proposes to use interval analysis for measurements, which makes it possible to assess only the aspects of uncertainty that can be assessed, and not all possible ones. We mean that the approach proposed by D. Dubois makes it possible to conduct analysis in a situation of "inaccuracy" and "uncertainty" and can be used to assess the probability of threats in information systems.

In the methodology, the base risk of the commands family is determined not by a fixed set of values provided by experts, but dynamically, based on a quantitative assessment of the "damage" that can be caused by the execution of commands of this specific family in an application environment (server). This approach, widely known as "user behavior," is considered the main way to detect anomalies, i.e., undesirable user actions [1], [4]. Universal operating systems can be considered as an application environment, as well as, for example, the automotive industry [12]. This approach ensures the scalability of the model and its adaptation to different infrastructures, severity levels, and threat profiles.

***Notation and definitions.*** Before introducing the model, we fix the objects it operates on. All of them are deterministic quantities computed from observed data. The only empirical inputs are the damage estimates and the execution counts.

- $\mathcal{C}$ is the finite set of normalized commands produced by the procedure of Section 2; $C \in \mathcal{C}$ is a single command, i.e., a base command together with its parameters and context parameters;
- $F$ is the set of command families, see (1);
- $\varphi$ is the family mapping, which assigns to every command exactly one family, $\varphi(C) = F_i$;
- $\mathcal{P}$ is the set of parameters (flags, paths and the context parameters of Table 1);
- $P(C) \subseteq \mathcal{P}$ is the parameters present in the command $C$, see (10);
- $L$ is the damage function; $L(F_i)$ is a point estimate of the expected damage of family $F_i$ obtained as described below, not a random variable;
- $R_0(F_i) \to [R_{\min}, R_{\max}]$ is the base risk of a command family, defined by (9);
- $k \to (0, \infty)$ is the coefficient function;
- $k_{F_i}(p)$ is the influence coefficient of parameter $p$ for family $F_i$, with $k_{F_i}(p) = 1$ whenever $p$ is not relevant for $F_i$;
- $f_s$ is the empirical execution count of a command on a server $s$ over an observation window, see (17);
- $Q$, $R^*$, $w_s$, $R_1$, $R_2$ are the structural score (11), the saturated risk (12), the rarity weight (19) and the local and global risk indicators (20)–(21), respectively. The complete model, combining all of the above in a single expression, is stated in (22) at the end of this section.

Suppose you have a set of command families:

$$F = \{F_1, F_2, \dots, F_m\} \quad (1)$$

For each command family $F_i \in F$ is estimated the expected "damage" $L(F_i)$, which can be expressed in monetary units or another quantitative metric reflecting the consequences of a breach of confidentiality, integrity, or availability.

Expected damage $L(F_i)$ can be estimated based on the following factors:

- incident statistics;
- scenario modeling;
- expert assessments;
- internal methods for assessing the damage inside the organization.

Thus, the underlying risk is based on a measurable value rather than a subjective ranking. Damage from different command families can vary significantly from each other:

- view of directories – almost insignificant damage (in turn, viewing can affect the collection of information);
- reconfiguration (change of configuration) – moderate damage;
- mass data deletion – significant damage.

If we use a linear scale, then small and medium values become indistinguishable. Therefore, the methodology is supplemented by a logarithmic transformation, which makes it possible to smooth out differences and maintain monotonicity.

To do this, let's introduce the auxiliary value U, calculated for each instruction family $F_i$:

$$\mathrm{U}(F_i) = \log_{10}(L(F_i) + 1), \quad (2)$$

where adding "one" to the expression ensures correctness provided that $L(F_i) = 0$.

Properties of an auxiliary value $\mathrm{U}(F_i)$:

- when $\mathrm{L}(F_i) = 0$ then $U(F_i) = 0$;
- exponential damage growth is converted into linear;
- differences of several orders of magnitude between the values become comparable.

After taking the logarithm, the values of $U(F_i)$ are reduced to the scale $[R_{min}, R_{max}]$, in this methodology, we use the scale $[0,100]$ throughout the paper.

For further exposition, let us define the extremes:

$$\mathrm{U}_{min} = \min_{F_i \in F} U(F_i), \quad (3)$$

$$\mathrm{U}_{max} = \max_{F_i \in F} U(F_i), \quad (4)$$

Then, the base risk value for the command family $F_i \in F$ is calculated using the logarithmic normalization formula [13]:

$$R_0(F_i) = A + B \cdot U(F_i), \quad (5)$$

where $A$ and $B$ are linear transformation parameters that ensure that the values are converted to the selected range [13]. The parameters $A$ and $B$ are fixed by the boundary conditions:

$$R_0(F_i) = R_{\min} \text{ if } U(F_i) = U_{\min} \quad (6)$$

$$R_0(F_i) = R_{\max} \text{ if } U(F_i) = U_{\max} \quad (7)$$

and for the calculation of $A$ and $B$ we get the system of equations:

$$\begin{cases} A + B \cdot U_{min} = R_{min} \\ A + B \cdot U_{max} = R_{max} \end{cases} \quad (8)$$

Having calculated $A$ and $B$, we can get the risk value:

$$R_0(F_i) = R_{min} + \frac{R_{max} - R_{min}}{U_{max} - U_{min}} \cdot (U(F_i) - U_{min}) \quad (9)$$

Properties of the expression (9) are:

- the monotonicity of the function is preserved;
- the scale is uniform over the logarithm of the damage $\mathrm{U}(F_i)$.

Formula (9) is well defined whenever $U_{\max} > U_{\min}$, i.e., whenever at least two command families differ in expected damage. In the degenerate case $U_{\max} = U_{\min}$, when all families have the same expected damage, we set $R_0(F_i) = R_{\max}$ for all $i$ by convention: no family is then less dangerous than another, and commands are differentiated by their parameters only.

Since $L(F_i)$ is expressed in a monetary or other quantitative unit, we note how (2) and (9) behave under a change of unit $L \to cL$, $c > 0$. The transformation (2) is not exactly invariant, because $\log_{10}(cL + 1) \neq \log_{10}(L + 1) + \log_{10} c$. However, for $L \gg 1$ the difference is a constant shift $\log_{10} c$ up to $O(1/L)$, and a constant shift of all $U(F_i)$ is removed by the min–max normalization (9). Hence $R_0$ is invariant under a change of the damage unit whenever the smallest damage $\min_i L(F_i)$ is large compared to 1 in that unit. The term "+1" in (2) serves only to handle the case $L(F_i) = 0$ and is negligible otherwise. In practice, the unit is chosen so that the smallest non-zero damage is at least of the order of $10^2$, which keeps the residual dependence on the unit within about one point of the 100-point scale.

The most important characteristic of the methodology is that the base risk is not fixed. It is recalculated anew for each application of the methodology, based on the following factors:

- up-to-date damage data;
- infrastructure features;
- criticality of the information systems;
- threat model.

At the same time:

- the methodology remains unchanged;
- only the input data changes;
- the methodology can be applied to different types and scales of infrastructure;
- the methodology can be adapted to a specific organization and context.

Thus, the methodology works in real time; if necessary, the initial parameters can be updated. The methodology does not impose universal risk values but provides a universal mechanism that automatically adjusts to the context, that is, to the behavior of the user and a specific information system (server).

After determining the base risk $R_0(F_i)$ of command family, the risk of a particular command is refined based on its structural and contextual parameters. Each parameter reflects a specific aspect of the command behavior that can amplify or lower potential damage.

For each parameter $p_j$, the influence coefficient $k_{F_i}(p_j)$, is provided, which quantitatively describes the change in the risk value if this parameter is available for a specific command family $\mathrm{F}_i \in F$. The explanation of the effect of the parameters on the risk value is given in 0.

Table 2. Influence on risk by parameters

| Value | Influence |
|---|---|
| $k_{F_i}(p_j) \in (0,1)$ | Decreases the risk |
| $k_{F_i}(p_j) = 1$ | Non influencing |
| $k_{F_i}(p_j) > 1$ | Increases the risk |

*Source: created by the authors*

The coefficient $k_{F_i}(p_j)$ is a factor reflecting the relative change in risk attributable to the parameter $p_j$. We adopt the modeling assumption that the contributions of different parameters are independent, i.e., the effect of a parameter does not depend on which other parameters are present in the same command. This is an assumption of the model rather than a consequence of it; it motivates the multiplicative form used below, and its limitations are discussed in Section 5.

Suppose command $C_k$ belongs to the command family $F_i$ and a set of parameters is given:

$$P(C_i) = \{p_1, p_2, \dots, p_n\}. \quad (10)$$

The set of parameters is the same for all command families; if a parameter is not relevant for a family, the corresponding coefficient equals 1. This avoids a separate parameter set for every command and keeps the model simple. We first define the structural score of a command as the product:

$$Q(C_i \mid p_1, \dots, p_s) = R_0\big(\varphi(C_i)\big) \cdot \prod_{t=1}^{s} k_{\varphi(C_i)}(p_t) \quad (11)$$

where $\varphi(C_i)$ denotes the family of $C_i$. The product (11) is unbounded above: a command of a family with $R_0 = R_{\max}$ executed with the parameters sudo, -r, -f and a path in /etc (Table 4, Appendix) yields $Q = 100 \cdot 10 \cdot 2 \cdot 5 \cdot 3 = 30\,000$, far outside the range $[R_{\min}, R_{\max}]$ fixed in (5). To map $Q$ back to the working scale while preserving the ranking it induces, we apply the hyperbolic saturation:

$$R^*(C_i \mid p_1, \dots, p_s) = R_{\max} \cdot \frac{Q}{Q + R_{\max}} \quad (12)$$

Transformation (12) is preferred to hard clipping $\min(R_{\max}, Q)$, which collapses all sufficiently dangerous commands to the same value and destroys their ordering, and to min–max renormalization over the observed command set, which would make the score of a command depend on which other commands happened to be observed. In (12), an unmodified command of the most dangerous family ($Q = R_{\max}$) receives $R^* = R_{\max}/2$, so that half of the scale remains available to separate aggravating parameters. A steeper or flatter saturation can be obtained from the generalization $R_{\max} \cdot Q^a/(Q^a + R_{\max}^a)$ with $a > 0$; we use $a = 1$ throughout.

The model (11)–(12) has the following properties; the proofs are elementary and follow from the sign and magnitude of the derivatives of (12).

*Proposition 1 (Boundedness).* If $R_0 \in [0, R_{\max}]$ and $k_F(p) > 0$ for all families and parameters, then $0 \le R^* < R_{\max}$.

*Proposition 2 (Monotonicity).* $R^*$ is strictly increasing in $R_0$ and in every coefficient $k_F(p_t)$. In particular, adding a parameter with $k > 1$ strictly increases $R^*$, and adding a parameter with $k \in (0,1)$ strictly decreases it.

*Proposition 3 (Order preservation).* Since (12) is a strictly increasing bijection of $[0, \infty)$ onto $[0, R_{\max})$, $R^*(C) \le R^*(C')$ if and only if $Q(C) \le Q(C')$. The saturation changes the scale but not the ranking of commands produced by (11).

*Proposition 4 (Sensitivity to coefficient uncertainty).*

$$\frac{\delta R^*}{\delta \ln k_F(p_t)} = \frac{R_{\max}^2 Q}{(Q + R_{\max})^2} \le \frac{R_{\max}}{4}$$

Hence, a relative error $\delta$ in any single coefficient changes $R^*$ by at most $(R_{\max}/4) \cdot |\ln(1+\delta)|$; for the 100-point scale, a 50 % error in any coefficient of Table 3 and Table 4 (Appendix) shifts the score by at most ≈10 points and a factor-of-two error by at most ≈17 points. The ranking of commands is therefore robust to moderate miscalibration of the coefficients, which is examined empirically in Section 4.

We note further tha+96t (12) is the same function for all infrastructures, information systems and organizations. Only the base risk $R_0$ and the coefficients $k$ are environment-specific, and the scalability of the methodology follows from the dynamic nature of $R_0$ described above.

## 3.1 Risk-decreasing Command Parameters

Some parameters of commands indicate that there is no undesirable impact. Examples are given in 0.

Table 3. Examples of risk-decreasing semantic tags

| Semantic tag | Flag spellings (commands) | $k_F$ | Description |
|---|---|---|---|
| help | --help, -h (most GNU utilities) | 0.25 | Prints usage information; no action is performed |
| version | --version, -V (most GNU utilities) | 0.25 | Prints the version of the command |
| dry run | --dry-run, -n (rsync, apt, make) | 0.5 | Simulates the action without applying changes |

*Source: created by the authors*

For such parameters (for all command families) the coefficient satisfies:

$$k_F(--help) \in (0,1) \quad (13)$$

For example, parameter "--help" (shows the help of the command) can be defined as:

$$k_F(--help) = 0.25 \quad (14)$$

Then, for the command `rm -help` risk calculation formula is defined as:

$$Q(\mathrm{rm} \mid --help) = R_0\big(\varphi(\mathrm{rm})\big) \cdot 0.25,$$
$$R^*(\mathrm{rm} \mid --help) = R_{\max} \cdot \frac{Q}{Q+R_{\max}} \quad (15)$$

for $R_0\big(\varphi(\mathrm{rm})\big) = 80$ this gives $Q = 20$ and $R^* \approx 16.7$.

The command remains a member of its family, but the action it performs in this form is limited to printing help information of the command ("--help" is for this purpose), so its structural score is reduced accordingly.

This is a heuristic, not a security guarantee: a non-standard program may attach side effects to "--help" or "--version", and a family whose commands are known to behave in this way should have the corresponding coefficient set to 1. The purpose of the risk-decreasing tags is to avoid the false positives that a pure anomaly detector would raise on such invocations, as discussed above.

## 3.2 Risk-increasing Command Parameters

Risk-increasing parameters include parameters that perform actions that are forced (without additional confirmation), propagated to all subdirectories (recursively), that indicate access to system parts of operating systems, or that escalate privileges when executed. Some of the risk-increasing parameters are given in 0 (Appendix).

For such parameters, the coefficient is determined:

$$k_F > 1 \quad (16)$$

The coefficients defined for command parameters $k_F$ have properties:

- clarify the base risk of the command $R_0(F_i)$ considering the parameters used;
- provide flexibility in the methodology;
- allow considering both "dangerous" (increasing risk) and safe (reducing risk) parameters;
- maintain the monotonicity of (11)–(12), see Proposition 2.

## 3.3 Coefficient Elicitation and Calibration

The base risk $R_0$ is derived from damage estimates, whereas the coefficients $k_F$ are assigned by a procedure external to the model. The values in Table 3 and Table 4 (Appendix) are illustrative and environment-specific rather than universal constants. This asymmetry is deliberate: damage is a property of the protected system and is measurable, whereas a coefficient expresses how a parameter changes the reach or the reversibility of an action and is therefore defined relative to the base risk of the command family. Three procedures for obtaining the coefficients are foreseen:

- *Expert elicitation* by pairwise comparison in the analytic hierarchy process (AHP) [14]. Experts compare the parameters pairwise on a ratio scale; the coefficient vector is obtained from the principal eigenvector of the comparison matrix, and judgments with a consistency ratio above 0.1 are rejected. The values of Table 4 (Appendix) correspond to the judgment that privilege escalation is twice as severe as forced execution, which in turn is 2.5 times as severe as recursion.
- *Supervised calibration* when labeled incidents are available. Because of the multiplicative form (11), $Q$ is linear in the logarithms of the coefficients, so that $k_F$ can be estimated by logistic regression of the incident label on the parameters, with $R_0$ as an offset; the estimates and their confidence intervals then replace the expert values.
- *Ordinal specification only*. When neither experts nor labels are available, the model requires only the ordering of the parameters and the coefficients $k_F$ (risk-increasing or risk-decreasing). By Proposition 4, the score obtained with any values consistent with that ordering differs from the score obtained with the tabulated values by a bounded amount, and by Proposition 3 the ranking depends only on the products of the coefficients.

In all three cases, the coefficients are re-estimated when the parameters, the coefficients, or the infrastructure change. The sensitivity of the final ranking to their values is examined in Section 4.

In the methodology, a mathematical model of the impact of a command on the information system is introduced, which transforms the user's action (executed command) in the operating system (i.e., a text record) into a value in the range from 0 to 100, characterizing the potential "danger" of the action performed by the user.

The saturated risk R* characterizes a command in isolation. The second component of the model accounts for how usual the command is in the observed environment, which is measured by the frequency of its execution on the server or across the infrastructure.

Frequencies are not interpreted as probabilities: no probability space over commands is assumed, and the quantities introduced below are empirical counts and deterministic weights derived from them. Logarithmic normalization of the counts is used to avoid the dominance of frequently called navigation and auxiliary commands in the final scale.

Unlike the approach [5], which considers any anomaly in user behavior as an information security incident, the methodology allows for taking into account additional parameters when performing actions by users to reduce false-positive values (deliberately false values), when the user can execute a potentially dangerous command, for example, with the "--version" parameter. Such actions (with risk-reducing parameters) in systems with UEBA (User Behavior Analysis) can cause an anomaly and the formation of an information security incident, which leads to false conclusions.

*Frequency-based weighting.* Let $T$ be an observation window (e.g., a rolling window of a fixed number of sessions or days) and $s$ a server.

For each command $C \in \mathcal{C}$, denote by $f_s(C)$ the number of executions of $C$ on server $s$ during $T$, and by $f_s^{\max} = \max_{C \in \mathcal{C}} f_s(C)$ the largest count in the window. The set $\mathcal{C}$ is the set of commands observed on any monitored server; a command that has never been observed on $s$ has $f_s(C) = 0$. In the following, $f$ is a deterministic empirical count, not a probability:

$$f_s(C),\ C \in \mathcal{C} \tag{17}$$

Command frequencies in administrative sessions are heavily skewed: a small number of navigation and inspection commands ("cd", "ls", "cat", "pwd") account for most executions. A linear normalization $\frac{f}{f^{\max}}$ would therefore assign nearly identical values to all but a few commands. To retain resolution in the tail of the distribution, we use the logarithm-normalized frequency:

$$\Pr_s(C) = \frac{\log(1+f_s(C))}{\log(1+f_s^{\max})} \tag{18}$$

with the convention $\Pr_s \equiv 0$ when $f_s^{\max} = 0$ (empty window). $\Pr_s(C) \in [0,1]$ is monotonically increasing in $f_s(C)$; the logarithm compresses the range of large counts so that commands with moderate frequency are distinguishable from rare ones (for $f^{\max} = 1000$, a command with $f = 50$ receives $\Pr \approx 0.57$ rather than $0.05$).

The purpose of frequency weighting in this methodology is not to estimate expected loss but to emphasize commands that are unusual for the observed environment. We therefore define the rarity weight:

$$w_s(C) = \varepsilon + (1-\varepsilon)\big(1 - \Pr_s(C)\big), \quad \varepsilon \in (0,1) \tag{19}$$

where $\varepsilon$ is a floor that prevents the most frequent command from being assigned zero risk. We define it as a constant in the paper and the value $\varepsilon = 0.1$.

The weight has the following properties:

- $w_s(C) \in [\varepsilon, 1]$ and is strictly decreasing in $f_s(C)$;
- a command not observed on $s$ ($f_s(C) = 0$) receives the maximum weight $w_s(C) = 1$, so novel commands are never suppressed;
- the ordering of commands by frequency is preserved (reversed) in the ordering by weight;
- $w_s$ is Lipschitz in $f_s$ with constant $\frac{(1-\varepsilon)}{\ln(1+f_s^{\max})}$, so small changes in the observed counts produce small changes in the weight;
  This is the stability notion of [15]; the Lipschitz property follows from the mean value theorem, since the derivative of the weight with respect to the count is bounded by this constant [16].
- if the observation window changes, $f_s$ and $f_s^{\max}$ are recomputed and (18)–(19) are applied unchanged.

Two instances of the weight are used: the local weight $w_1(C)$ computed from the counts on a single server, and the global weight $w_2(C)$ computed from the aggregated counts over all monitored servers. The local and global risk indicators of a command $C$ with parameters $p_1, \dots, p_s$ are:

$$R_1(C \mid p_1, \dots, p_s) = R^*(C \mid p_1, \dots, p_s) \cdot w_1(C) \tag{20}$$

$$R_2(C \mid p_1, \dots, p_s) = R^*(C \mid p_1, \dots, p_s) \cdot w_2(C) \tag{21}$$

Since $w \in [\varepsilon, 1]$, both indicators lie in $[0, R^*]$ and are non-decreasing in $R^*$ and non-increasing in the corresponding frequency.

The pair $(R_1, R_2)$ separates two situations that a single score cannot: $R_1 < R_2$ indicates an action that is routine for the given server but unusual for the infrastructure as a whole – typical of a legitimate

user with a specialized role – whereas $R_1 \approx R_2$ with a high value indicates an action that is unusual everywhere. This is the mechanism by which the methodology reduces false positives relative to approaches that treat any deviation from the population baseline as an incident [5].

The complete definition of the risk quantity combining (9), (11)–(12) and (19)–(21) is given in Figure 1.

In a single expression, the risk of command $C$ with parameters $P(C)$ executed on server $s$ is:

$$R_s(C) = R_{\max} \cdot \frac{Q(C)}{Q(C) + R_{\max}} \cdot w_s(C),$$
$$Q(C) = R_0\big(\varphi(C)\big) \cdot \prod_{p \in P(C)} k_{\varphi(C)}(p) \quad (22)$$

where $s$ ranges over a single server (giving $R_1$) or the whole set of monitored servers (giving $R_2$). The properties established above transfer to (22) directly: $R_s(C) \in [0, R_{\max})$ by Proposition 1 and (19), $R_s$ is non-decreasing in $L\big(\varphi(C)\big)$ and in every coefficient by Proposition 2, non-increasing in the execution count $f_s(C)$, and Lipschitz in $f_s(C)$ with constant $\frac{R_{\max}(1-\varepsilon)}{\ln(1+f_s^{\max})}$ .

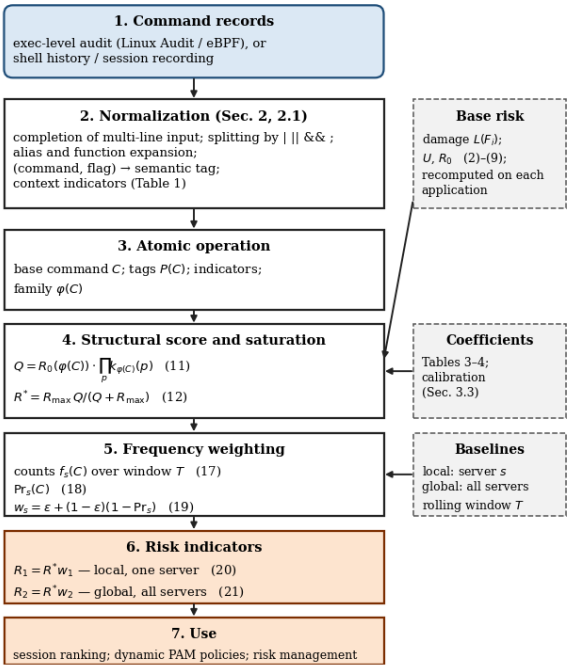


Fig. 1: Processing pipeline of the methodology from raw command records to the risk indicators $R_1$ and $R_2$

*Source: created by the authors*

# 4 Experimental Evaluation

This section illustrates the model on a worked example with the coefficients of Table 3 and Table 4 (Appendix) defines the evaluation protocol applied to command traces from production servers.

## 4.1 Worked Example

Consider three command families with base risk $R_0$ = 5 (routine operations), 40 (service management) and 80 (data deletion), an observation window on one server with $f^{\max} = 1000$, and $\varepsilon = 0.1$.

Table 5 lists five commands with their structural score Q (11), saturated risk $R^*$ (12), local count $f_s$, rarity weight $w_s$ (19), $R_{\max} = 100$ and local indicator $R_1$ (20).

Table 5. Worked example of the risk calculation

| Command (family; tags) | $R_0$ | Q | $R^*$ | $f$ | $w_s$ | $R_1$ |
|---|---|---|---|---|---|---|
| ls -la (routine) | 5 | 5 | 4.76 | 1000 | 0.100 | 0.48 |
| systemctl restart nginx (service management) | 40 | 40 | 28.57 | 50 | 0.488 | 13.94 |
| rm --help (deletion; help 0.25) | 80 | 20 | 16.67 | 2 | 0.857 | 14.28 |
| rm -rf /etc/nginx (deletion; recursive 2, force 5, system path 3) | 80 | 2 400 | 96.00 | 1 | 0.910 | 87.33 |
| sudo rm -rf /etc/nginx (as above + privilege escalation 10) | 80 | 24 000 | 99.59 | 0 | 1.000 | 99.59 |

*Source: created by the authors*

Four observations follow from Table 5.

First, the unbounded product Q of the last command (24 000) is mapped by (12) onto the working scale without collapsing the two dangerous commands to the same value, as hard clipping would do.

Second, the same base command rm receives 14.28 with the "—help" parameter and 87.33 with the recursive, force, and system-path; i.e., the parameters – not the command name – determine the score, which is the property that a name-based rule cannot provide.

Third, the command never observed on the server receives the maximum weight $w_s$ = 1 and the highest indicator, whereas the routine navigation command executed a thousand times is suppressed to 0.48 despite a non-zero base risk.

Fourth, halving or doubling every coefficient of Table 3 and Table 4 (Appendix) moves the indicators of the two dangerous commands within the interval [68, 100] and never changes their rank relative to the other commands, whereas the order of the two low-scoring commands, whose indicators

differ by 0.3 points, is not stable; this is the expected behavior under Proposition 4, since the bound on the sensitivity guarantees the ranking only where scores are separated.

## 4.2 Evaluation on Production Command Traces

*Dataset*. The evaluation uses the shell histories of seven administrators on seven production servers (web, mail, and infrastructure hosts, abbreviated S1…S7 in the paper) collected over six months; four servers run Ubuntu 22.04, two run Ubuntu 24.04, and one runs Ubuntu 26.04. The records are of the secondary type of Section 2.1: shell history without timestamps and without the working directory. The 5 480 history lines were split at pipelines and command lists into 5 890 atomic operations, normalized as described in Section 2 (alias expansion with the Ubuntu defaults, sudo and wrapper removal, tag assignment by the command-specific dictionary, context indicators of Table 1) and grouped into 260 distinct keys (base command, tag set, family) over the whole fleet and 499 distinct (server, key) pairs. The distribution is as skewed as assumed in Section 3: navigation and inspection commands account for 64.1 % of all operations, the five most frequent keys (ls, cd, cp, systemctl, and less on system paths) account for 55.4 %, and 36.5 % of the (server, key) pairs were observed exactly once on their server. The tags occur 659 times (system path), 156 (recursive), 96 (force), 49 (privilege escalation), 30 (help/version), 16 (auto-confirm), and once each (interpreter inline code, encoded input). Since the history contains no timestamps, the observation window T is the whole period, and the decay of the weight of a newly introduced command over successive windows could not be measured; since it contains no working directory, relative paths into system directories (e.g. editing dovecot.passwd after cd /etc/dovecot) are not recognized by the system-path indicator – both are limitations of the secondary source that an exec-level record removes.

*Parameters.* Eleven command families were used; their relative damage estimates $L(F_i)$, assigned by the authors on a conditional scale for the purpose of the experiment, the resulting base risk $R_0$ by (9) and the share of operations per family are given in Table 6. The coefficients of Table 3 and Table 4 (Appendix) were used without adjustment; $R_{\max} = 100$ and $\varepsilon = 0.1$. The scoring pipeline is a Python program; the traces, the dictionary, and the program are available from the authors.

Table 6. Command families, relative damage, base risk and share of operations in the traces

| Command family | $L$ | $R_0$ | Share, % |
|---|---|---|---|
| navigation and inspection | 1 | 0.0 | 64.1 |
| other (unrecognized) | 5 | 28.0 | 1.8 |
| file operations | 15 | 53.0 | 13.1 |
| network communications | 20 | 60.0 | 0.4 |
| package management | 30 | 69.9 | 1.8 |
| application administration | 35 | 73.7 | 4.6 |
| service management | 40 | 77.0 | 6.0 |
| script execution | 45 | 79.9 | 0.9 |
| configuration change | 50 | 82.6 | 3.4 |
| user and access management | 70 | 91.0 | 0.2 |
| data deletion (incl. log truncation) | 100 | 100.0 | 3.8 |

*Source: created by the authors*

Table 7.Ten operations from the traces with their local and global risk indicators

| Server (S + number): operation (family; tags) | Q | $R^*$ | $f_1$ / $f_2$ | $w_1$ / $w_2$ | $R_1$ | $R_2$ |
|---|---|---|---|---|---|---|
| S1: sudo tee /etc/apt/sources.list.d/ … (file operations; privilege escalation, system path) | 1 590.6 | 94.1 | 1 / 3 | 0.88 / 0.83 | 82.7 | 77.8 |
| S3: sudo su (user and access management; privilege escalation) | 910.1 | 90.1 | 1 / 4 | 0.89 / 0.80 | 80.3 | 72.0 |
| S3: sudo vi /etc/raddb/mods-available/ldap (configuration change; privilege escalation, system path) | 2 477.3 | 96.1 | 2 / 2 | 0.83 / 0.86 | 79.6 | 83.0 |
| S2: rm -rf ./2023-11* (data deletion; recursive, force) | 1 000.0 | 90.9 | 1 / 80 | 0.87 / 0.45 | 79.4 | 41.1 |
| S7: xargs sed -i 's#<?php /**/ eval(base64_decode( …' (file operations; encoded input) | 265.1 | 72.6 | 1 / 1 | 0.87 / 0.91 | 63.1 | 66.3 |
| S5: crontab -e (configuration change) | 82.6 | 45.2 | 3 / 13 | 0.78 / 0.67 | 35.2 | 30.3 |
| S2: echo "" > btmp.1 (data deletion, log truncation) | 100.0 | 50.0 | 11 / 23 | 0.55 / 0.60 | 27.3 | 30.2 |
| S2: systemctl restart exim4 (service management) | 77.0 | 43.5 | 11 / 295 | 0.55 / 0.29 | 23.8 | 12.6 |
| S2: doveadm --help (application administration; help/version) | 18.4 | 15.6 | 3 / 3 | 0.75 / 0.83 | 11.6 | 12.9 |
| S2: ls -l (navigation and inspection) | 0.0 | 0.0 | 138 / 1364 | 0.10 / 0.10 | 0.0 | 0.0 |

*Source: created by the authors*

*Ranking.* Nineteen of the twenty (server, key) pairs with the highest local indicator $R_1$ are operations executed at most twice on their server; they combine privilege escalation with a system path (sudo tee /etc/apt/…, sudo vi /etc/raddb/…, sudo ln -s … /usr/bin/…), open a root shell (sudo

su), or delete recursively without confirmation (rm -rf …). Table 7 shows ten operations taken from the dataset and selected to span the scale; the rows are ordered by $R_1$.

Table 7 illustrates the behavior of the components. The operation rm -rf ./2023-11* was executed once on the server S2, but recursive forced deletion is the most frequent dangerous key in the fleet (80 executions), hence $R_1 = 79.4$ but $R_2 = 41.1$ : unusual for the server, routine for the infrastructure. The reverse case, $R_2 > R_1$ , is exemplified outside the table by "nano /etc/postfix/main.cf" on server S4, executed 43 times locally and 50 times globally ($R_1$ = 22.1, $R_2$= 36.3): routine for the mail administrator, unusual elsewhere – the "legitimate user" pattern that motivated the two indicators. The log truncation "echo "" > btmp.1", repeated eleven times on server S2, is halved by the frequency weight; the model treats repeated log wiping as routine, which is the intended output for a user freeing disk space and the wrong one for an intruder covering tracks – the metric cannot distinguish the two without labels, see Section 5. The in-place edit of all PHP files with a pattern containing "eval(base64_decode(…))" triggers the encoded-input parameter; the command was a malware clean-up rather than an attack, but an unseen "sed -i" over the whole web root is an operation that merits attention, so the score of 63.1 is not a false positive of the metric even though it is one of the lexical indicators. Finally, "ls -l" with 1 364 executions and "doveadm --help" with the help tag show the two mechanisms that keep routine and informational operations at the bottom of the scale, for the least damaging family $R_0 = 0$ under the min–max normalization (9), so navigation commands score zero regardless of frequency.

*Sensitivity and baselines.* Table 8 reports the rank correlation between the reference ranking of the 499 (server, key) pairs by $R_1$ and the rankings obtained after the coefficients and damage estimates, and the correlation with three alternative scores. Perturbing any single coefficient by up to ±50% or any single damage estimate by ±50% leaves the ranking essentially unchanged ($\rho \geq 0.99$), and perturbing all of them jointly by random factors between 0.5 and 2 keeps ρ at 0.955 in the worst of 200 runs, in agreement with Proposition 4. The top-20 set is most sensitive to the coefficient of privilege escalation (overlap 80 % when it is halved), which is expected since it is the largest coefficient. The comparison with the alternative scores shows what each component contributes: the structural score alone reproduces the ranking with ρ = 0.962 but replaces 30% of the top-20 set, because it keeps repeatedly executed dangerous operations at the top ("rm -rf *" executed 42 times on server S3 has $R^* = 90.9$ but $R_1 = 37.5$); the frequency-only score, which corresponds to a UEBA-style baseline [5], has ρ = 0.318 and 15% overlap, because it ranks typos and one-off inspection commands together with dangerous ones. The global indicator agrees with the local one at ρ = 0.974; of the 499 pairs, 200 have $R_1 < R_2$, 134 have $R_1 > R_2$ and 165 (mostly pairs with $R^* = 0$) are equal, and the difference exceeds 20 points in four cases, all of them single recursive deletions or copies of system configuration files that are frequent elsewhere on all servers.

Table 8. Rank correlation with the reference ranking by $R_1$ under perturbations and for alternative scores

| Perturbation / alternative score | ρ (Spearman) | τ (Kendall) | Top-20 overlap |
|---|---|---|---|
| one coefficient of Table 3 and Table 4 (Appendix), ×0.5 … ×1.5 (32 runs), worst case | 0.993 | 0.970 | 80 % |
| one damage estimate $L(F_i)$ , ×0.5 and ×1.5 (22 runs), worst case | 0.990 | 0.959 | 95 % |
| all coefficients and damages jointly, random factors in [0.5, 2] (200 runs), mean | 0.989 | 0.936 | 88 % |
| the same, worst case | 0.955 | 0.851 | 65 % |
| global indicator $R_2$ instead of $R_1$ | 0.974 | 0.887 | 75 % |
| structural score $R^*$ alone (no frequency component) | 0.962 | 0.872 | 70 % |
| rarity weight $w_1$ alone (frequency-only, UEBA-style baseline) | 0.318 | 0.247 | 15 % |

*Source: created by the authors*

The evaluation is a ranking evaluation: the traces carry no incident labels, so precision, ROC, and precision–recall curves are not reported. Obtaining labels by administrator review of the top-ranked operations, and repeating the evaluation on exec-level records with timestamps, are the next steps.

# 5 Limitations and Future Work

The following limitations of the present work are acknowledged.

- The normalization procedure of Section 2 is a lexical normalizer; it does not preserve the security-relevant semantics of arbitrary shell input, and the evasion techniques listed in Section 2.1 are mitigated by the rarity weight and the context parameters rather than eliminated (Section 2.1).

- The mapping from (command, flag) pairs to semantic tags is a dictionary that has to be maintained; flags absent from the dictionary are treated as neutral, which underestimates the risk of unlisted dangerous options.
- The coefficients of Table 3 and Table 4 (Appendix) are expert-specified values; Proposition 4 bounds the effect of their miscalibration, but the calibration procedures of Section 3.3 have not yet been compared on labeled data.
- The independence of parameter contributions assumed in (11) is a modeling assumption; interactions between tags (e.g., force together with recursive) are captured only through the product of their coefficients.
- The frequency component depends on the choice of the observation window T and on the floor $\varepsilon$; A window that is too short inflates the weight of legitimate periodic maintenance; a window that is too long delays the detection of a change in behavior.
- Risk-decreasing tags are heuristics and not security guarantees, as noted in Section 3.1.
- The evaluation of Section 4 is a ranking evaluation; a classification-quality evaluation (ROC, precision–recall) requires labeled incident data, which is the subject of future work, together with sequence-level modeling of sessions and data-driven updating of the base risk.

# 6 Conclusion

The methodology proposed in this paper allows applying additional command parameters and calculating risk values for each user action. Thus, the methodology provides a possibility to assess the risks of user actions in Linux-based operating systems, allowing the use of the assessment data in dynamic information security risk management systems [7], as well as in various scenarios for automating access provisioning and control. For privileged access management (PAM) systems the indicators $R_1$ and $R_2$ can serve as an input to dynamic access-control policies, for example by requiring step-up confirmation for commands whose indicator exceeds a threshold. The evidence for this use is at present limited to the ranking evaluation of Section 4; the evaluation on labeled incident data and the integration with a PAM product remain future work.

**Declaration of Generative AI and AI-assisted Technologies in the Writing Process**

During the preparation of this work, the author(s) utilized artificial intelligence (AI) technologies solely for the purpose of linguistic refinement and grammatical correction. Following the application of these tools, the author(s) comprehensively reviewed, verified, and edited the resulting text. The author(s) maintain full accountability and responsibility for the accuracy, integrity, and final content of the publication.

*References:*

[1] Kim J, Park M, Kim H, Cho S, Kang P. Insider Threat Detection Based on User Behavior Modeling and Anomaly Detection Algorithms. Applied Sciences. 2019; 9(19):4018. https://doi.org/10.3390/app9194018.

[2] Atlam, Hany & Azad, Muhammad & Alassafi, Madini & Alshdadi, Abdulrahman & Alhomoud, Ahmed. (2020). Risk-Based Access Control Model: A Systematic Literature Review. Future Internet. 12. 24. https://doi.org/10.3390/fi12060103.

[3] Riaz Ahmed Shaikh, Kamel Adi, Luigi Logrippo, Dynamic risk-based decision methods for access control systems, Computers & Security, Volume 31, Issue 4, 2012, Pages 447-464, ISSN: 0167-4048, https://doi.org/10.1016/j.cose.2012.02.006.

[4] Jose Fuentes, Ines Ortega-Fernandez, Nora M. Villanueva, Marta Sestelo. Cybersecurity threat detection based on a UEBA framework using Deep Autoencoders[J]. AIMS Mathematics, 2025, 10(10): 23496-23517. https://doi.org/10.3934/math.20251043.

[5] *(text in Russian)* Nashivochnikov N., Ischukov D., Identification of deviations in behavioral patterns of users of corporate information resources using topological features. // Russia, Cybersecurity questions. – 2023. #4(56). p.70–81. https://doi.org/10.21681/2311-3456-2023-4-12-22.

[6] Pavlos Cheimonidis, Konstantinos Rantos, A novel proactive and dynamic cyber risk assessment methodology, Computers & Security, Volume 154, 2025, 104439, ISSN: 0167-4048, https://doi.org/10.1016/j.cose.2025.104439.

[7] Pengchao Yao, Qiang Yang, Wenhai Wang, Dynamic cross-layer security risk assessment and mitigation for cyber-physical power systems, Reliability Engineering & System

Safety, Volume 261, 2025, 111027, ISSN: 0951-8320, https://doi.org/10.1016/j.ress.2025.111027.

[8] Hertzog R., Mas R. The Debian Administrator's Handbook: Debian Wheezy from Discovery to Mastery, [Online]. https://www.debian.org/doc/manuals/debian-handbook (Accessed Date: March 10, 2026).

[9] Linux Audit userspace project, [Online]. https://github.com/linux-audit/audit-userspace (Accessed Date: March 10, 2026).

[10] IEEE and The Open Group. The Open Group Base Specifications Issue 7, 2018 Edition (IEEE Std 1003.1-2017), Chapter 2: Shell Command Language, [Online]. https://pubs.opengroup.org/onlinepubs/9699919799/utilities/V3_chap02.html (Accessed Date: January 15, 2026).

[11] Dubois D. Possibility theory : An approach to computerized processing of uncertainty /; With the collab. of Henri Farreny et al. ; Transl. by E. F. Harding. - New York; London: Plenum press, Cop. 1988. - XVI, 263 p. ISBN: 0-306-42520-3.

[12] Merola, F.; Bernardeschi, C.; Lami, G. A Risk Assessment Framework Based on Fuzzy Logic for Automotive Systems. Safety 2024, 10, 41. https://doi.org/10.3390/safety10020041.

[13] Mosteller F.J., Tukey J.W., Data Analysis and Regression: A Second Course in Statistics. Reading, MA, USA: Addison-Wesley, 1977. ISBN: 020104854X.

[14] Saaty T.L. The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation. New York, NY, USA: McGraw-Hill, 1980. ISBN: 0070543712.

[15] Bousquet O., Elisseeff A. Stability and generalization. Journal of Machine Learning Research, Volume 2, 2002, Pages 499–526. https://doi.org/10.1162/153244302760200704

[16] Rudin W. Principles of Mathematical Analysis, 3rd ed. New York, NY, USA: McGraw-Hill, 1976. ISBN: 0070856133.

**Contribution of Individual Authors to the Creation of a Scientific Article (Ghostwriting Policy)**

- Dmitry A. Belyavskiy formulated the problem, developed the methodology, implemented the scoring pipeline, and carried out the experiments of Section 4. Artem
- S. Kabanov and Alexey B. Los contributed to the formal analysis of the model and to the analysis of the results. All authors participated in writing and revising the manuscript.

**Sources of Funding for Research Presented in a Scientific Article or Scientific Article Itself**

No funding was received for conducting this study.

**Conflict of Interest**

The authors have no conflicts of interest to declare that are relevant to the content of this article.

**Creative Commons Attribution License 4.0 (Attribution 4.0 International, CC BY 4.0)**

This article is published under the terms of the Creative Commons Attribution License 4.0
https://creativecommons.org/licenses/by/4.0/deed.en_US

# APPENDIX

Table 4. Examples of risk-increasing semantic tags

| Semantic tag | Flag spellings (commands) | $k_F$ | Description |
|---|---|---|---|
| recursive | -r, -R, --recursive<br>(rm, cp, chmod, chown) | 2 | Propagates the action to all subdirectories |
| force | -f, --force<br>(rm, cp, mv, git) | 5 | Performs the action without confirmation, overriding protections |
| system path | /etc, /bin, /usr, /var<br>(any command; Table 1) | 3 | Targets system directories |
| auto-confirm | -y, --yes,<br>--assume-yes<br>(apt, yum, dnf) | 2 | Answers all prompts affirmatively |
| interpreter inline code | bash -c, sh -c, python -c, perl -e<br>(Table 1) | 3 | Executes code passed as an argument |
| encoded input | base64 -d, xxd -r, eval<br>(Table 1) | 5 | Executes decoded or dynamically built input |
| privilege escalation | sudo, su, doas | 10 | Executes the command with elevated privileges |

*Source: created by the authors*

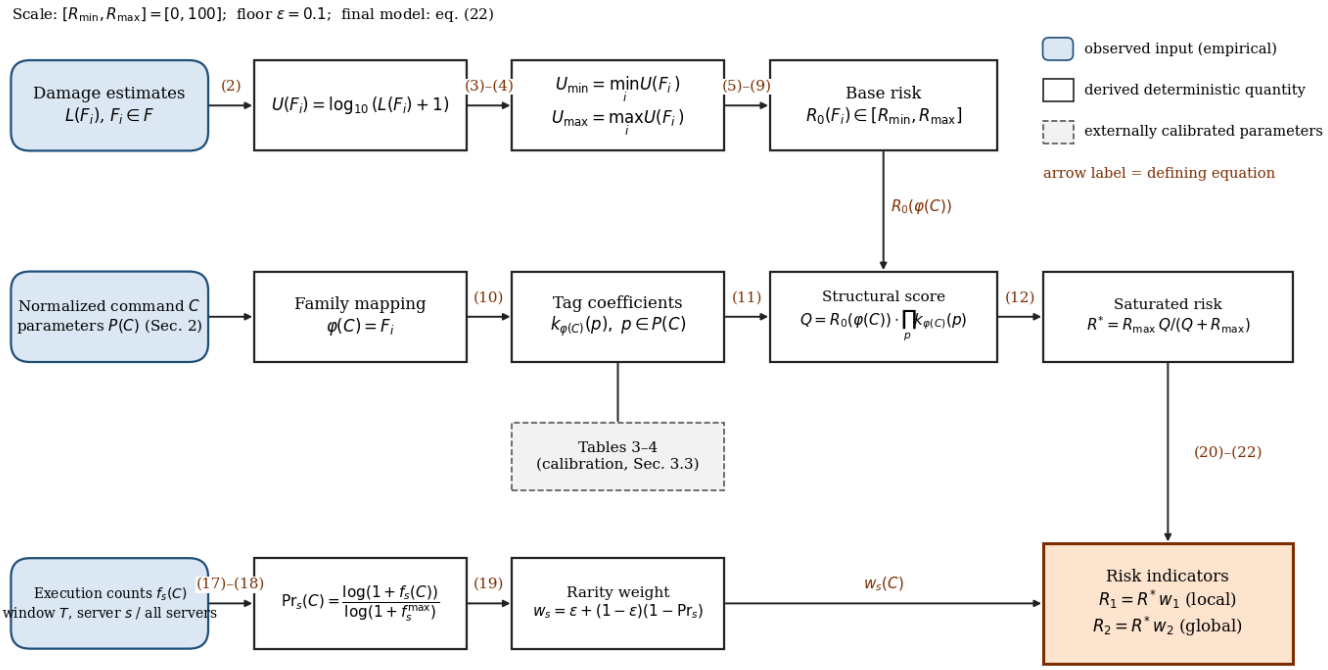


Fig. 2: Dependency diagram of the risk model:
observed inputs, derived quantities and the equations that define them (Section 3)
*Source: created by the authors*