# UserArmor: An extension for AppArmor

Pierpaolo Sestito[1,*]

[1]University of Calabria, Department of Mathematics and Computer Science

**Abstract**

Answer Set Programming is a declarative programming paradigm designed for solving complex combinatorial problems and is widely used in artificial intelligence, knowledge representation and automated reasoning. ASP can be used to develop web-based AI applications, where an ASP solver is executed in the backend server. While this approach is functional, it introduces serious security vulnerabilities if not properly implemented. This work introduces UserArmor, an extension of AppArmor, with the aim of hardening ASP solvers and other Linux processes on a per-user basis.

**Keywords**

ASP Solver Hardening, Logic-based Profile Generation, Policy Reasoning, User-level Access Control

## 1. Introduction and problem description

The state-of-the-art solver CLINGO[1] is one of the most popular Answer Set Programming, offering a powerful engine that can be extended with Python v3.12 and Lua v5.4 scripts to enhance reasoning capabilities. ASP can be used to develop web-based AI applications, and this is a common scenario for researchers aiming to showcase their AI-powered tools that solve hard combinatorial tasks using ASP[2, 3, 4]. This approach is functional, but it also exposes the system to remote code execution (RCE) vulnerabilities. CLINGO supports @-terms, which allow interpreted functions to be evaluated dynamically and these functions (Python or Lua), meaning that CLINGO can execute arbitrary Python or Lua code. Malicious users could input a crafted ASP program containing a @-term that executes system commands, leading to arbitrary code execution on the backend server. Focusing on Python, for example, a user could submit an ASP program like the following:

```
#script(python)
import subprocess
def rce(cmd):
    return subprocess.check_output(["sh", "-c", cmd.string], text=True)
#end.
out(@rce("whoami")).
```

The above program can be easily adapted to compromise the security. A robust solution is to contain the execution of CLINGO by restricting the permissions of the process running the web-app.

AppArmor presents one possible solution. AppArmor[5] is a Mandatory Access Control (MAC) framework for Linux[6, 7], based on security profiles that restrict access to system resources[8, 9]. Profiles define strict rules for files, directories, networking and other critical resources[9, 10], enforced at the kernel level and independent of application logic. AppArmor supports per-process or per-application policies, but lacks user-specific enforcement. This makes it inadequate for multi-user scenarios, where different users running the same process may require distinct restrictions. This means that every instance of CLINGO on the backend server would be restricted, including those run by legitimate users (e.g, via SSH). So by applying a global AppArmor profile to CLINGO, all users on the system would be restricted in the same way, including users who are supposed to retain full access.

This paper introduces *UserArmor*, an extension of AppArmor that adds user-level policy granularity and tag-based inheritance mechanisms. User-level granularity is implemented via a hierarchical

*ICLP DC 2025: 21st Doctoral Consortium on Logic Programming, September 2025, Rende, Italy.*

*Corresponding author.

pierpaolo.sestito@outlook.it (P. Sestito)

structure: each application has a base profile, and each user is linked to a nested subprofile. UserArmor associates each application with a directory of per-user subprofiles, stored in separate files for modularity. These are merged into a single mappings file, included via `#include` in the main policy. At load time, the kernel links user-specific profiles under names like `confined_app//user_name`. Inheritance is achieved through a tag-based system using `#@selectable` and `#@select` keywords, which enable reuse of rules without duplication. Their usage is detailed in Section 4.

We demonstrate how UserArmor can be applied to *harden web-deployed ASP solvers like CLINGO*, mitigating remote code execution risks by confining the execution environment of specific users (e.g., `www-data`) without affecting others. For instance, by defining a dedicated security profile for the web service user, CLINGO can be restricted with *no file system access, no network access, and a minimal execution environment*.

Looking ahead, we plan to further extend UserArmor by integrating *Answer Set Programming techniques* not only for the *automatic generation of user-specific policies*, but also for reasoning about internal implementation concerns such as *automated handling of inheritance and profile consistency*. In particular, we envision converting audit logs and trace events (e.g., from `Auditd` or `SystemTap`) into ASP facts, from which logic-based rules can derive abstract properties (e.g., inferring `network_acceve agss("dns")`) and enforce security constraints declaratively.

A UserArmor profile for `www-data` could look like this:

```
profile www-data {
    # Deny access to the entire filesystem except for necessary paths
    deny / rwx,
    /usr/lib/** rm,
    # Allow read access only to the directory where ASP encodings are stored
    /var/www/clingo_input/** r,
    # No network access
    deny network,
    # Deny execution of system commands
    deny capability sys_admin,
    deny capability setuid,
    deny capability setgid,
}
```

With this policy, even if an attacker injects Python or Lua code, CLINGO cannot access critical files, execute system commands, or connect to the internet, effectively neutralizing the RCE attack.

## 2. Background and overview of the existing literature

In AppArmor, processes can be associated with *profiles* that control access to system resources. Profiles are typically stored in `/etc/apparmor.d/`, named after the executable's absolute path (with `/` replaced by `.`). A profile has the structure:

```
profile NAME /ABSOLUTE/PATH {
  RULES -> define access permissions
  SUBPROFILES -> specify rules for subprocesses
}
```

In the main profile, `profile NAME` can be omitted. Rules can apply to:

**Files**: `/ABSOLUTE/PATH FLAGS`, where `FLAGS` include `r` (read), `w` (write), `x` (execute), and `ix` (execute under current profile).

**Capabilities**: `capability CAPABILITIES`, where `CAPABILITIES` include kernel-level privileges (e.g., `setuid`).

**Networking**: `network TYPE`, with `TYPE` specifying communication types (e.g., `inet` or `inet6`).

Any rule can be prefixed with `deny` to block access.

Key AppArmor tools include `aa-genprof` (interactive profile generation), `aa-logprof` (profile updates from audit logs), `apparmor_parser` (profile verification and loading), and `aa-complain`, `aa-enforce`, `aa-disable` (to switch enforcement modes).

AppArmor supports rule reuse through *abstraction*, which are dedicated files containing reusable sets of rules, included with: `#include <file>` or, conditionally: `include if exists <file>`.

Paths are relative to `/etc/apparmor.d/`. Missing files trigger an error unless the conditional form is used.

These files contain common permissions and access patterns for widely used applications or subsystems, such as *bash*, *nameservice*, *X11*.

### 2.1. Literature review

The closest work to ours is Paranoid Penguin [9], which improves AppArmor usability by providing a graphical interface for profile creation and management. This approach reduces configuration errors and makes AppArmor more accessible, but it does not alter the enforcement model or introduce new security features. UserArmor, instead, extends AppArmor's model with fine-grained access control for multi-user environments and a tag-based inheritance system that avoids rule duplication.

Related research also investigates dynamic profiling, where policies adapt at runtime. Some proposals offload profile generation to the cloud for centralized control [11, 12], while others apply updates locally to enable immediate response [13]. Container security is another active area: tools such as Auditd and SystemTap have been employed to generate Docker profiles dynamically [14, 15], later integrated into unified frameworks [11] and enhanced with live updates [16] and Kubernetes support [17]. Unlike these dynamic approaches, which react to observed behavior, UserArmor enforces static, user-aware policies that offer low overhead and predictable results. It further supports resource limits (CPU, memory, file handles), strengthening its role within a layered defense strategy.

Traditional UNIX-based systems already provide basic user-level access control mechanisms, including file ownership and permission bits, group memberships, and POSIX Access Control Lists (ACLs). While effective for managing file and resource access, these mechanisms are not integrated with kernel-level security modules like AppArmor and lack process-level confinement or modular policy enforcement. Resource limits can be applied per user through PAM configurations (e.g., *limits.conf*) [18, 19, 20], but these settings are coarse-grained and disconnected from application-specific security requirements. UserArmor instead offers a unified and scalable approach to multi-user confinement.

Role-Based Access Control (RBAC) is also available in Linux through SELinux, which provides a powerful yet complex security model based on types, roles, and domains [21, 10, 22]. Although SELinux enables role-based confinement, it is usually deployed with predefined, system-wide policies and lacks straightforward mechanisms for assigning tailored restrictions to individual users running the same application. Crafting custom SELinux policies requires in-depth knowledge of its security architecture [23], making it less suitable for scenarios that demand lightweight, user-specific confinement. UserArmor fills this gap by offering a practical solution for managing multi-user security policies without requiring global policy reconfiguration.

## 3. Goal of the research

The goal of this research is to provide an extension of AppArmor that overcomes the lack of user-level granularity and inheritance mechanisms. This motivated the study and implementation of UserArmor.

**Example 1.** *Consider an application executed by two users,* user1 *and* user2*, which writes to a user-specific log file. The most restrictive AppArmor profile that allows the application to function would include:*

```
/var/log/my_confined_app/user1.log rw,
/var/log/my_confined_app/user2.log rw,
```

*Further restrictions would break application logic, as each user needs access to their own log file. This creates a vulnerability: a bug could let* user1 *access* user2*'s file, violating confidentiality and integrity. To prevent this, we introduce per-user subprofiles:*

```
profile user1 {
    ...
    /var/log/my_confined_app/user1.log rw,
}
```

*And* `profile user2` *with 'rw' permission on* `/var/log/my_confined_app/user2.log` *file. When the application is run by* `user1`*, the* `user1` *subprofile is enforces, preventing any access to* `user2.log` *at the kernel level, even in the presence of application bugs.* ■

While profiles can be nested, subprofiles define separate rules for subprocesses and do not inherit from the parent, as different privileges may be required.

AppArmor supports rule reuse via *abstractions*, included with `#include`. However, abstractions are *all-or-nothing* and cannot adapt based on conditions such as the user identity. This makes policy management complex and error-prone in user-specific or frequently changing environments.

**Example 2.** *Consider a Bash application,* `my_confined_app`*, that uses* `cat` *to read a config file and requires network access. It can be run by two* sudo *users, but only one needs administrative privileges. As in Example 1, each user must access their own log file. A standard AppArmor profile for this scenario would be:*

```
#include <tunables/global>
/usr/bin/my_confined_app {
    profile user1 {
        #include <abstractions/bash>
        /etc/my_confined_app.conf r,
        /usr/bin/cat ix,
        network inet,
        /var/log/my_confined_app/user1.log rw,
    }

    profile user2 {
        #SAME RULES
        /var/log/my_confined_app/user2.log rw,
    }
}
```

*Here,* abstractions *include common Bash permissions (e.g.,* `.bash_profile`*,* `.bash_rc`*,* `.profile`*), but many rules are duplicated across subprofiles. UserArmor addresses this with a tag-based inheritance mechanism that simplifies rule reuse:*

```
#include <tunables/global>
/usr/bin/my_confined_app {
    #OTHER RULES
    #@selectable{adm}  capability sys_admin,
    #@selectable{net}  network inet,
    profile user1 {
/var/log/my_confined_app/user1.log rw,
    }
    profile user2 {
/var/log/my_confined_app/user2.log rw,
    }
}
```

*Their usage is detailed in Section 4.* ■

## 4. Current status of the research

UserArmor uses a tag system represented with a comment-based syntax, described below. Rules (and blocks) are associated with aliases (i.e., identifying strings).

**Base rules.** Untagged rules are considered essential and are automatically inherited by all subprofiles.

**Selectable rules.** Rules starting with a tag `#@selectable{ALIAS}` are not inherited automatically (and are not part of the main profile), but can be included in the subprofiles via their `ALIAS`.

**Selectable blocks.** A syntax similar to the previous one can be applied to rule blocks:

```
#@selectable{ALIAS}
#     RULES
#@end
```

**Removable rules.** Rules ending with a tag `#@removable{ALIAS}` are inherited in subprofiles unless explicitly removed by their `ALIAS`.

**Subprofile inheritance.** Subprofiles can select rules and blocks using the following tag: `#@select: ALIASES` where `ALIASES` is a space-separated list of aliases. Selected rules and blocks are added to the basic rules. Untagged rules are added at the begin of each subprofile, unless their alias is listed in the following tag: `#@remove: ALIASES`

The tag system applies to profile files stored in a directory specific to the confined application. Each user has a profile file, and a central `mappings` file includes them all. This `mappings` file is then included in the main profile under `/etc/apparmor.d/`. AppArmor loads subprofiles as `confined_app//user_name`, allowing UserArmor to select the correct one based on the current user.

**Example 3 (Continuing Example 2).** *The scenario before is modeled with the profile file `/etc/apparmor.d/usr.bin.my_confined_app`:*

```
    #include <tunables/global>
/usr/bin/my_confined_app {
    #include <abstractions/bash>
    /etc/my_confined_app.conf r,
    /usr/bin/cat ix,
    #@selectable{adm}  capability sys_admin,
    #@selectable{net}  network inet,
    include if exists <.usr.bin.my_confined_app/mappings>
}
```

*The directory `/etc/apparmor.d/.usr.bin.my_confined_app` contains:*

- *`user1`, associated to the first user, with the following content:*

  ```
  profile user1 {
      #@select: adm net
      /var/log/my_confined_app/user1.log rw,
  }
  ```

- *`user2`, associated with the second user, has same structure as `user1`, but selects only the `net` tag, limiting access accordingly.*
- *`mappings`, including the above files with the expansion of the selected permissions (e.g user1):*

  ```
  profile user1 {
      #@select: adm net
      capability sys_admin,
      network inet,
      /var/log/my_confined_app/user1.log rw,
  }
  ```

*UserArmor automates this setup by inserting the* `include if exists <.usr.bin.my_confined_app/mappings>` *directive into the application profile and generating* `/etc/apparmor.d/.usr.bin.my_confined_app/mappings`*. Untagged rules are treated as essential and inherited automatically.* ■

UserArmor includes three command-line utilities: `ua-generate`, `ua-enforce`, and `ua-exec`.

`ua-generate` creates the required directory and one subprofile per user, preserving existing files (Root privileges are required). `ua-enforce` inserts the `include if exists` directive and generates the `mappings` file based on tag selection (Also requires root). `ua-exec` selects and applies the correct subprofile using `aa-exec`. Usable by any user, but `aa-exec` should be executable only by root or members of the `userarmor` group. Users outside the group will fall back to the base profile; group members use their dedicated subprofile.

**Example 4 (Continuing Example 3).** *Starting from the base profile* `/etc/apparmor.d/usr.bin.my_confined_app`*, the following command generates the directory structure and empty subprofiles:*

```
$ sudo ua-generate /usr/bin/my_confined_application --subprofiles=user1,user2
```

*After editing the subprofiles (as in Example 3), the administrator runs:*

```
$ sudo ua-enforce /usr/bin/my_confined_application
```

*This command generates the* `mappings` *file, ensures the* `include if exists` *directive is present, and enforces the policy via AppArmor. To run the application,* `user1` *uses:*

```
$ ua-exec /usr/bin/my_confined_app arg1 arg2 ...
```

*Running the app without* `ua-exec`*:*

```
$ my_confined_app arg1 arg2 ...
```

*would fall back to the base profile, with no access to* `/var/log/my_confined_app/user1.log`*.* ■

## 5. Preliminary results

We provide scripts for four scenarios: (i) Unconfined CLINGO, where the web app is vulnerable to RCE and system compromise; (ii) AppArmor-confined, which prevents RCE but restricts all users, including those accessing via SSH; (iii) UserArmor-confined, which blocks RCE for the web app while allowing unrestricted SSH use of CLINGO; and (iv) Bubblewrap sandboxed, which prevents RCE but requires a fresh isolated environment for each execution.

The script files are available online (https://github.com/pierpaolosestito-dev/ASPArmor; accessed 13 February 2025). We ran an experiment on a 13th Gen Intel(R) Core(TM) i7-1360P @ 2.2 GHz CPU with 32 GB RAM using a simple Bash script:

```
cat /etc/my_confined_app.conf  >> /var/log/my_confined_app/$USER.log
```

We repeatedly executed CLINGO on the program: `number(1..10000)`

UserArmor introduced negligible overhead. For example, after 50 calls, execution times were: Unconfined: 0.99s, AppArmor: 1.01s, UserArmor: 1.08s, Bubblewrap: 1.32s. Performance remained consistent across larger batches, with UserArmor consistently outperforming Bubblewrap.

In summary, securing CLINGO in web environments is essential due to the risk posed by @-terms, which enable RCE. Bubblewrap provides isolation but adds overhead and requires workflow changes. UserArmor offers a lighter, kernel-level alternative: it restricts CLINGO for the web user (`www-data`), while leaving other users unaffected.

For completeness, we note that CLINGO—like MiniZinc [24] and Nemo [25, 26]—can also run in the browser via WebAssembly. A notable example is ASP Chef [27, 28], a low-code ASP environment integrating multiple languages and frameworks [29, 30].

# 6. Open issues and expected achievements

## 6.1. Logic-based automatic policy generation

A central line of future work will be the integration of declarative reasoning techniques to support the automatic generation of UserArmor profiles. Building on existing approaches that use tools such as Auditd and SystemTap to monitor system events, we plan to convert these events into facts that can be processed by an ASP solver.

From these low-level facts, additional rules can derive higher-level properties (e.g. grouping raw TCP connection events into abstract facts such as `network_access("dns")`, which can then be constrained by declarative policies.

**Example 5.** *A concrete case is a log entry from Auditd like:*

```
type=SYSCALL msg=audit(...): exe="/usr/bin/clingo" comm="python3" syscall=2
    file="/etc/passwd"
```

*which can be translated into the fact*

```
access("clingo","python3","/etc/passwd",read).
```

*and combined with a constraint such as*

```
:- access(_, _, File, _), sensitive(File).
```

*to ensure that no policy allows access to sensitive files. Similarly, a network event like*

```
network_request("clingo","8.8.8.8").
```

*can be constrained with*

```
:- network_request(_, Host), Host != "trusted.example.org".
```

The solver will search for combinations of permissions that are both safe and minimal. High-level constraints - such as ”deny all network access except to domain X” - will be expressed as declarative rules. In this framework, logic is not just a description of the observed events but an active reasoning tool that: synthesizes profiles by automatically selecting permissions, verifies that generated policies satisfy safety constraints, and explains every decision by linking it to rules and facts.

This reasoning-based extension will make UserArmor a hybrid framework where static enforcement and logic-based reasoning cooperate to improve security in multi-user environments.

## 6.2. Additional tools and practical challenges

Beyond the integration of ASP-based reasoning, future development will also address the creation of administrative tools for managing user-specific profiles, including the ability to rename or delete them. It will further investigate the scalability of reasoning when large execution traces are collected and explore interactive workflows in which administrators can refine or approve the policies generated automatically. These directions aim to turn UserArmor into a practical yet research-oriented platform that combines secure Linux confinement with the flexibility of logic programming.

# Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT-4o for grammar and spelling check. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the publication’s content.

## References


[1] M. Gebser, R. Kaminski, T. Schaub, Complex optimization in answer set programming, Theory Pract. Log. Program. 11 (2011) 821–839.

[2] T. Eiter, G. Ianni, R. Schindlauer, H. Tompits, A uniform integration of higher-order reasoning and external evaluations in answer-set programming, in: L. P. Kaelbling, A. Saffiotti (Eds.), IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005, Professional Book Center, 2005, pp. 90–96. URL: http://ijcai.org/Proceedings/05/Papers/1353.pdf.

[3] F. Calimeri, S. Germano, E. Palermiti, K. Reale, F. Ricca, Developing ASP programs with ASPIDE and loide, Künstliche Intell. 32 (2018) 185–186. URL: https://doi.org/10.1007/s13218-018-0534-z. doi:10.1007/S13218-018-0534-Z.

[4] F. Calimeri, D. Fuscà, S. Germano, S. Perri, J. Zangari, Fostering the use of declarative formalisms for real-world applications: The embasp framework, New Gener. Comput. 37 (2019) 29–65. URL: https://doi.org/10.1007/s00354-018-0046-2. doi:10.1007/S00354-018-0046-2.

[5] C. Cowan, Securing linux systems with apparmor, DEF CON 15 (2007) 15–26.

[6] Y. Jiang, C. Lin, H. Yin, Z. Tan, Security analysis of mandatory access control model, in: 2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No. 04CH37583), volume 6, IEEE, 2004, pp. 5013–5018.

[7] S. Osborn, Mandatory access control and role-based access control revisited, in: Proceedings of the second ACM workshop on Role-based access control, 1997, pp. 31–40.

[8] T. Ecarot, S. Dussault, A. Souid, L. Lavoie, J. Ethier, Apparmor for health data access control: Assessing risks and benefits, in: L. Boubchir, E. Benkhelifa, Y. Jararweh, I. Saleh (Eds.), 7th International Conference on Internet of Things: Systems, Management and Security, IOTSMS 2020, Virtual Event, France, December 14-16, 2020, IEEE, 2020, pp. 1–7. URL: https://doi.org/10.1109/IOTSMS52051.2020.9340206. doi:10.1109/IOTSMS52051.2020.9340206.

[9] M. Bauer, Paranoid penguin: an introduction to novell apparmor, Linux Journal 2006 (2006) 13.

[10] H. Chen, N. Li, Z. Mao, Analyzing and comparing the protection quality of security enhanced operating systems., in: NDSS, volume 9, 2009.

[11] H. Zhu, C. Gehrmann, Apparmor profile generator as a cloud service, in: M. Helfert, D. Ferguson, C. Pahl (Eds.), Proceedings of the 11th International Conference on Cloud Computing and Services Science, CLOSER 2021, Online Streaming, April 28-30, 2021, SCITEPRESS, 2021, pp. 45–55. URL: https://doi.org/10.5220/0010434100450055. doi:10.5220/0010434100450055.

[12] H. Zhu, C. Gehrmann, P. Roth, Access security policy generation for containers as a cloud service, SN Computer Science 4 (2023) 748.

[13] Y. Li, C. Huang, L. Yuan, Y. Ding, H. Cheng, Aspgen: an automatic security policy generating framework for apparmor, in: J. Hu, G. Min, N. Georgalas, Z. Zhao, F. Hao, W. Miao (Eds.), IEEE International Conference on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking, ISPA/BDCloud/SocialCom/SustainCom 2020, Exeter, United Kingdom, December 17-19, 2020, IEEE, 2020, pp. 392–400. URL: https://doi.org/10.1109/ISPA-BDCloud-SocialCom-SustainCom51426.2020.00075. doi:10.1109/ISPA-BDCLOUD-SOCIALCOM-SUSTAINCOM51426.2020.00075.

[14] M. Mattetti, A. Shulman-Peleg, Y. Allouche, A. Corradi, S. Dolev, L. Foschini, Securing the infrastructure and the workloads of linux containers, in: 2015 IEEE Conference on Communications and Network Security (CNS), 2015, pp. 559–567. doi:10.1109/CNS.2015.7346869.

[15] F. Loukidis-Andreou, I. Giannakopoulos, K. Doka, N. Koziris, Docker-sec: A fully automated container security enhancement mechanism, in: 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), IEEE, 2018, pp. 1561–1564.

[16] C. Huang, K. Wang, Y. Li, J. Li, Q. Liao, Aspgen-d: Automatically generating fine-grained apparmor policies for docker, in: IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking, ISPA/BDCloud/SocialCom/Sustain-

Com 2022, Melbourne, Australia, December 17-19, 2022, IEEE, 2022, pp. 822–829. URL: https://doi.org/10.1109/ISPA-BDCloud-SocialCom-SustainCom57177.2022.00110. doi:10.1109/ISPA-BDCLOUD-SOCIALCOM-SUSTAINCOM57177.2022.00110.

[17] H. Zhu, C. Gehrmann, Kub-sec, an automatic kubernetes cluster apparmor profile generation engine, in: 14th International Conference on COMmunication Systems & NETworkS, COMSNETS 2022, Bangalore, India, January 4-8, 2022, IEEE, 2022, pp. 129–137. URL: https://doi.org/10.1109/COMSNETS53615.2022.9668504. doi:10.1109/COMSNETS53615.2022.9668504.

[18] A. G. Morgan, Pluggable authentication modules for linux: An implementation of a user-authentication api, Linux Journal 1997 (1997) 1–es.

[19] K. Geisshirt, Pluggable authentication modules, Birmingham, UK: Packt Publishing 105 (2007).

[20] V. Samar, Unified login with pluggable authentication modules (pam), in: Proceedings of the 3rd ACM conference on Computer and communications security, 1996, pp. 1–10.

[21] Z. C. Schreuders, T. McGill, C. Payne, Empowering end users to confine their own applications: The results of a usability study comparing selinux, apparmor, and fbac-lsm, ACM Trans. Inf. Syst. Secur. 14 (2011). URL: https://doi.org/10.1145/2019599.2019604. doi:10.1145/2019599.2019604.

[22] C. Shepherd, K. Markantonakis, Operating system controls, in: Trusted Execution Environments, Springer, 2024, pp. 33–53.

[23] S. Smalley, Configuring the selinux policy, NAI Labs Rep (2002) 02–007.

[24] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, G. Tack, MiniZinc: Towards a standard CP modelling language, in: C. Bessiere (Ed.), CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings, volume 4741 of *LNCS*, Springer, 2007, pp. 529–543. doi:10.1007/978-3-540-74970-7_38.

[25] A. Ivliev, L. Gerlach, S. Meusel, J. Steinberg, M. Krötzsch, Nemo: A scalable and versatile datalog engine, in: M. Alviano, M. Lanzinger (Eds.), Proceedings 5th International Workshop on the Resurgence of Datalog in Academia and Industry (Datalog-2.0 2024) co-located with the 17th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2024), Dallas, Texas, USA, October 11, 2024, volume 3801 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2024, pp. 43–47. URL: https://ceur-ws.org/Vol-3801/short3.pdf.

[26] A. Ivliev, L. Gerlach, S. Meusel, J. Steinberg, M. Krötzsch, Nemo: Your friendly and versatile rule reasoning toolkit, in: P. Marquis, M. Ortiz, M. Pagnucco (Eds.), Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning, KR 2024, Hanoi, Vietnam. November 2-8, 2024, 2024. URL: https://doi.org/10.24963/kr.2024/70. doi:10.24963/KR.2024/70.

[27] M. Alviano, D. Cirimele, L. A. R. Reiners, Introducing ASP recipes and ASP chef, in: J. Arias, S. Batsakis, W. Faber, G. Gupta, F. Pacenza, E. Papadakis, L. Robaldo, K. Rückschloß, E. Salazar, Z. G. Saribatur, I. Tachmazidis, F. Weitkämper, A. Z. Wyner (Eds.), Proceedings of the International Conference on Logic Programming 2023 Workshops co-located with the 39th International Conference on Logic Programming (ICLP 2023), London, United Kingdom, July 9th and 10th, 2023, volume 3437 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023. URL: https://ceur-ws.org/Vol-3437/paper4ASPOCP.pdf.

[28] M. Alviano, L. A. R. Reiners, ASP chef: Draw and expand, in: P. Marquis, M. Ortiz, M. Pagnucco (Eds.), Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning, KR 2024, Hanoi, Vietnam. November 2-8, 2024, 2024. URL: https://doi.org/10.24963/kr.2024/68. doi:10.24963/KR.2024/68.

[29] M. Alviano, L. A. R. Reiners, Integrating minizinc with ASP chef: Browser-based constraint programming for education and prototyping, in: C. Dodaro, G. Gupta, M. V. Martinez (Eds.), Logic Programming and Nonmonotonic Reasoning - 17th International Conference, LPNMR 2024, Dallas, TX, USA, October 11-14, 2024, Proceedings, volume 15245 of *Lecture Notes in Computer Science*, Springer, 2024, pp. 174–186. URL: https://doi.org/10.1007/978-3-031-74209-5_14. doi:10.1007/978-3-031-74209-5_14.

[30] M. Alviano, P. Guarasci, L. A. R. Reiners, I. R. Vasile, Integrating structured declarative language (SDL) into ASP chef, in: C. Dodaro, G. Gupta, M. V. Martinez (Eds.), Logic Programming and Nonmonotonic Reasoning - 17th International Conference, LPNMR 2024, Dallas, TX, USA, October 11-14, 2024, Proceedings, volume 15245 of *Lecture Notes in Computer Science*, Springer, 2024, pp.

387–392. URL: https://doi.org/10.1007/978-3-031-74209-5_29. doi:10.1007/978-3-031-74209-5_29.