Ernesto Damiani
Peng Liu (Eds.)

Data and Applications Security XX

LNCS 4127

20th Annual IFIP WG 11.3 Working Conference on
Data and Applications Security
Sophia Antipolis, France, July/August 2006, Proceedings

ifip

Ernesto Damiani Peng Liu (Eds.)

Data and Applications Security XX

20th Annual IFIP WG 11.3 Working Conference on
Data and Applications Security
Sophia Antipolis, France, July 31–August 2, 2006
Proceedings

Springer

Volume Editors

Ernesto Damiani
University of Milan
Department of Information Technology
Via Bramante 65, 26013 Crema, Italy
E-mail: damiani@dti.unimi.it

Peng Liu
Pennsylvania State University
College of Information Sciences and Technology
313G IST Building, University Park, PA 16802, USA
E-mail: pliu@ist.psu.edu

Library of Congress Control Number: 2006929592

CR Subject Classification (1998): E.3, D.4.6, C.2, F.2.1, J.1, K.6.5

LNCS Sublibrary: SL 3 – Information Systems and Application, incl. Internet/Web and HCI

ISSN 0302-9743
ISBN-10 3-540-36796-9 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-36796-3 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11805588 06/3142 5 4 3 2 1 0

Lecture Notes in Computer Science 4127

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

Preface

For 20 years, the IFIP WG 11.3 Working Conference on Data and Applications Security (DBSEC) has been a major forum for presenting original research results, practical experiences, and innovative ideas in data and applications security. Looking back, it is difficult not to appreciate the full extent of the change that has occurred in our field. Once considered afterthoughts in systems and application design, data protection, privacy and trust have become the key problems of our day. This central role of security in the information society has however brought increased responsibilities to the research community. Today practitioners and researchers alike need to find new ways to cope with the increasing scale and complexity of the security problems that must be solved on the global information infrastructure. Like the previous conference, the 20th DBSEC has proved to be up to this challenge.

DBSEC 2006 received 56 submissions, out of which the program committee selected 22 high-quality papers covering a number of diverse research topics such as access control, privacy, and identity management. We are glad to see that the final program contains a well-balanced mix of theoretical results and practical prototype systems, many of them converging and building off each other. Also, the DBSEC program includes a number of papers on new, emerging aspects of security research.

Putting together a top-level conference like DBSEC is always a team effort. We would like to thank a number of friends and colleagues for their valuable help and support, including our General Chair Andreas Schaad, our Publicity Chair Soon Ae Chun, IFIP WG 11.3 Chair Pierangela Samarati, all our program committee members, and, above all, the researchers who chose DBSEC as the forum to which to submit their work. In addition, we would like to thank SAP for sponsoring this conference.

August 2006 Ernesto Damiani, Peng Liu

Organization

General Chair

Andreas Schaad	SAP Labs, France

Program Chairs

Ernesto Damiani	Università degli Studi di Milano, Italy
Peng Liu	Penn State University, USA

Publicity Chair

Soon Ae Chun	City University of New York, USA

IFIP WG 11.3 Chair

Pierangela Samarati	Università degli Studi di Milano, Italy

Program Committee

Gail-Joon Ahn	University of North Carolina at Charlotte, USA
Anne Anderson	Sun Microsystems, USA
Vijay Atluri	Rutgers University, USA
Sabrina De Capitani di Vimercati	Università degli Studi di Milano, Italy
Soon Ae Chun	City University of New York, USA
Chris Clifton	Purdue University, USA
Steve Demurjian	University of Connecticut, USA
Csilla Farkas	University of South Carolina, USA
Eduardo Fernandez-Medina	Univ. of Castilla-La Mancha, Spain
Simon Foley	University College Cork, Ireland
Qijun Gu	Texas State University, USA
Ehud Gudes	Ben-Gurion University, Israel
Sushil Jajodia	George Mason University, USA
Carl Landwehr	University of Maryland, USA
Tsau Young Lin	San Jose State University, USA
Patrick McDaniel	Pennsylvania State University, USA
Sharad Mehrotra	University of California Irvine, USA
Mukesh K. Mohania	IBM Research Labs, India
Ravi Mukkamala	Old Dominion University, USA

Peng Ning	North Carolina State University, USA
Sylvia Osborn	University of Western Ontario, Canada
Brajendra Panda	University of Arkansas, USA
Joon S. Park	Syracuse University, USA
Indrakshi Ray	Colorado State University, USA
Indrajit Ray	Colorado State University, USA
Pierangela Samarati	Università degli Studi di Milano, Italy
Andreas Schaad	SAP Labs, France
Sujeet Shenoi	University of Tulsa, USA
David Spooner	Rensselaer Polytechnic Institute, USA
Bhavani Thuraisingham	University of Texas Dallas, USA
T.C. Ting	University of Connecticut, USA
Duminda Wijesekera	George Mason University, USA
Meng Yu	Monmouth University, USA
Ting Yu	North Carolina State University, USA

Sponsoring Institutions

SAP Labs, Sophia Antipolis, France

Lecture Notes in Computer Science

For information about Vols. 1–3991

please contact your bookseller or Springer

Vol. 4127: E. Damiani, P. Liu (Eds.), Data and Applications Security XX. X, 319 pages. 2006.

Vol. 4090: S. Spaccapietra, K. Aberer, P. Cudré-Mauroux (Eds.), Journal on Data Semantics VI. XI, 211 pages. 2006.

Vol. 4079: S. Etalle, M. Truszczyński (Eds.), Logic Programming. XIV, 474 pages. 2006.

Vol. 4077: M.-S. Kim, K. Shimada (Eds.), Advances in Geometric Modeling and Processing. XVI, 696 pages. 2006.

Vol. 4076: F. Hess, S. Pauli, M. Pohst (Eds.), Algorithmic Number Theory. X, 599 pages. 2006.

Vol. 4075: U. Leser, F. Naumann, B. Eckman (Eds.), Data Integration in the Life Sciences. XI, 298 pages. 2006. (Sublibrary LNBI).

Vol. 4074: M. Burmester, A. Yasinsac (Eds.), Secure Mobile Ad-hoc Networks and Sensors. X, 193 pages. 2006.

Vol. 4073: A. Butz, B. Fisher, A. Krüger, P. Olivier (Eds.), Smart Graphics. XI, 263 pages. 2006.

Vol. 4072: M. Harders, G. Székely (Eds.), Biomedical Simulation. XI, 216 pages. 2006.

Vol. 4071: H. Sundaram, M. Naphade, J.R. Smith, Y. Rui (Eds.), Image and Video Retrieval. XII, 547 pages. 2006.

Vol. 4069: F.J. Perales, R.B. Fisher (Eds.), Articulated Motion and Deformable Objects. XV, 526 pages. 2006.

Vol. 4068: H. Schärfe, P. Hitzler, P. Øhrstrøm (Eds.), Conceptual Structures: Inspiration and Application. XI, 455 pages. 2006. (Sublibrary LNAI).

Vol. 4067: D. Thomas (Ed.), ECOOP 2006 – Object-Oriented Programming. XIV, 527 pages. 2006.

Vol. 4066: A. Rensink, J. Warmer (Eds.), Model Driven Architecture – Foundations and Applications. XII, 392 pages. 2006.

Vol. 4065: P. Perner (Ed.), Advances in Data Mining. XI, 592 pages. 2006. (Sublibrary LNAI).

Vol. 4064: R. Büschkes, P. Laskov (Eds.), Detection of Intrusions and Malware & Vulnerability Assessment. X, 195 pages. 2006.

Vol. 4063: I. Gorton, G.T. Heineman, I. Crnkovic, H.W. Schmidt, J.A. Stafford, C.A. Szyperski, K. Wallnau (Eds.), Component-Based Software Engineering. XI, 394 pages. 2006.

Vol. 4062: G. Wang, J.F. Peters, A. Skowron, Y. Yao (Eds.), Rough Sets and Knowledge Technology. XX, 810 pages. 2006. (Sublibrary LNAI).

Vol. 4061: K. Miesenberger, J. Klaus, W. Zagler, A. Karshmer (Eds.), Computers Helping People with Special Needs. XXIX, 1356 pages. 2006.

Vol. 4060: K. Futatsugi, J.-P. Jouannaud, J. Meseguer (Eds.), Algebra, Meaning and Computation. XXXVIII, 643 pages. 2006.

Vol. 4059: L. Arge, R. Freivalds (Eds.), Algorithm Theory – SWAT 2006. XII, 436 pages. 2006.

Vol. 4058: L.M. Batten, R. Safavi-Naini (Eds.), Information Security and Privacy. XII, 446 pages. 2006.

Vol. 4057: J.P. W. Pluim, B. Likar, F.A. Gerritsen (Eds.), Biomedical Image Registration. XII, 324 pages. 2006.

Vol. 4056: P. Flocchini, L. Gąsieniec (Eds.), Structural Information and Communication Complexity. X, 357 pages. 2006.

Vol. 4055: J. Lee, J. Shim, S.-g. Lee, C. Bussler, S. Shim (Eds.), Data Engineering Issues in E-Commerce and Services. IX, 290 pages. 2006.

Vol. 4054: A. Horváth, M. Telek (Eds.), Formal Methods and Stochastic Models for Performance Evaluation. VIII, 239 pages. 2006.

Vol. 4053: M. Ikeda, K.D. Ashley, T.-W. Chan (Eds.), Intelligent Tutoring Systems. XXVI, 821 pages. 2006.

Vol. 4052: M. Bugliesi, B. Preneel, V. Sassone, I. Wegener (Eds.), Automata, Languages and Programming, Part II. XXIV, 603 pages. 2006.

Vol. 4051: M. Bugliesi, B. Preneel, V. Sassone, I. Wegener (Eds.), Automata, Languages and Programming, Part I. XXIII, 729 pages. 2006.

Vol. 4049: S. Parsons, N. Maudet, P. Moraitis, I. Rahwan (Eds.), Argumentation in Multi-Agent Systems. XIV, 313 pages. 2006. (Sublibrary LNAI).

Vol. 4048: L. Goble, J.-J.C.. Meyer (Eds.), Deontic Logic and Artificial Normative Systems. X, 273 pages. 2006. (Sublibrary LNAI).

Vol. 4047: M. Robshaw (Ed.), Fast Software Encryption. XI, 434 pages. 2006.

Vol. 4046: S.M. Astley, M. Brady, C. Rose, R. Zwiggelaar (Eds.), Digital Mammography. XVI, 654 pages. 2006.

Vol. 4045: D. Barker-Plummer, R. Cox, N. Swoboda (Eds.), Diagrammatic Representation and Inference. XII, 301 pages. 2006. (Sublibrary LNAI).

Vol. 4044: P. Abrahamsson, M. Marchesi, G. Succi (Eds.), Extreme Programming and Agile Processes in Software Engineering. XII, 230 pages. 2006.

Vol. 4043: A.S. Atzeni, A. Lioy (Eds.), Public Key Infrastructure. XI, 261 pages. 2006.

Vol. 4042: D. Bell, J. Hong (Eds.), Flexible and Efficient Information Handling. XVI, 296 pages. 2006.

Vol. 4041: S.-W. Cheng, C.K. Poon (Eds.), Algorithmic Aspects in Information and Management. XI, 395 pages. 2006.

Vol. 4040: R. Reulke, U. Eckardt, B. Flach, U. Knauer, K. Polthier (Eds.), Combinatorial Image Analysis. XII, 482 pages. 2006.

Vol. 4039: M. Morisio (Ed.), Reuse of Off-the-Shelf Components. XIII, 444 pages. 2006.

Vol. 4038: P. Ciancarini, H. Wiklicky (Eds.), Coordination Models and Languages. VIII, 299 pages. 2006.

Vol. 4037: R. Gorrieri, H. Wehrheim (Eds.), Formal Methods for Open Object-Based Distributed Systems. XVII, 474 pages. 2006.

Vol. 4036: O. H. Ibarra, Z. Dang (Eds.), Developments in Language Theory. XII, 456 pages. 2006.

Vol. 4035: T. Nishita, Q. Peng, H.-P. Seidel (Eds.), Advances in Computer Graphics. XX, 771 pages. 2006.

Vol. 4034: J. Münch, M. Vierimaa (Eds.), Product-Focused Software Process Improvement. XVII, 474 pages. 2006.

Vol. 4033: B. Stiller, P. Reichl, B. Tuffin (Eds.), Performability Has its Price. X, 103 pages. 2006.

Vol. 4032: O. Etzion, T. Kuflik, A. Motro (Eds.), Next Generation Information Technologies and Systems. XIII, 365 pages. 2006.

Vol. 4031: M. Ali, R. Dapoigny (Eds.), Innovations in Applied Artificial Intelligence. XXIII, 1353 pages. 2006. (Sublibrary LNAI).

Vol. 4029: L. Rutkowski, R. Tadeusiewicz, L.A. Zadeh, J. Zurada (Eds.), Artificial Intelligence and Soft Computing – ICAISC 2006. XXI, 1235 pages. 2006. (Sublibrary LNAI).

Vol. 4027: H.L. Larsen, G. Pasi, D. Ortiz-Arroyo, T. Andreasen, H. Christiansen (Eds.), Flexible Query Answering Systems. XVIII, 714 pages. 2006. (Sublibrary LNAI).

Vol. 4026: P.B. Gibbons, T. Abdelzaher, J. Aspnes, R. Rao (Eds.), Distributed Computing in Sensor Systems. XIV, 566 pages. 2006.

Vol. 4025: F. Eliassen, A. Montresor (Eds.), Distributed Applications and Interoperable Systems. XI, 355 pages. 2006.

Vol. 4024: S. Donatelli, P. S. Thiagarajan (Eds.), Petri Nets and Other Models of Concurrency - ICATPN 2006. XI, 441 pages. 2006.

Vol. 4021: E. André, L. Dybkjær, W. Minker, H. Neumann, M. Weber (Eds.), Perception and Interactive Technologies. XI, 217 pages. 2006. (Sublibrary LNAI).

Vol. 4020: A. Bredenfeld, A. Jacoff, I. Noda, Y. Takahashi (Eds.), RoboCup 2005: Robot Soccer World Cup IX. XVII, 727 pages. 2006. (Sublibrary LNAI).

Vol. 4019: M. Johnson, V. Vene (Eds.), Algebraic Methodology and Software Technology. XI, 389 pages. 2006.

Vol. 4018: V. Wade, H. Ashman, B. Smyth (Eds.), Adaptive Hypermedia and Adaptive Web-Based Systems. XVI, 474 pages. 2006.

Vol. 4017: S. Vassiliadis, S. Wong, T.D. Hämäläinen (Eds.), Embedded Computer Systems: Architectures, Modeling, and Simulation. XV, 492 pages. 2006.

Vol. 4016: J.X. Yu, M. Kitsuregawa, H.V. Leong (Eds.), Advances in Web-Age Information Management. XVII, 606 pages. 2006.

Vol. 4014: T. Uustalu (Ed.), Mathematics of Program Construction. X, 455 pages. 2006.

Vol. 4013: L. Lamontagne, M. Marchand (Eds.), Advances in Artificial Intelligence. XIII, 564 pages. 2006. (Sublibrary LNAI).

Vol. 4012: T. Washio, A. Sakurai, K. Nakajima, H. Takeda, S. Tojo, M. Yokoo (Eds.), New Frontiers in Artificial Intelligence. XIII, 484 pages. 2006. (Sublibrary LNAI).

Vol. 4011: Y. Sure, J. Domingue (Eds.), The Semantic Web: Research and Applications. XIX, 726 pages. 2006.

Vol. 4010: S. Dunne, B. Stoddart (Eds.), Unifying Theories of Programming. VIII, 257 pages. 2006.

Vol. 4009: M. Lewenstein, G. Valiente (Eds.), Combinatorial Pattern Matching. XII, 414 pages. 2006.

Vol. 4008: J.C. Augusto, C.D. Nugent (Eds.), Designing Smart Homes. XI, 183 pages. 2006. (Sublibrary LNAI).

Vol. 4007: C. Àlvarez, M. Serna (Eds.), Experimental Algorithms. XI, 329 pages. 2006.

Vol. 4006: L.M. Pinho, M. González Harbour (Eds.), Reliable Software Technologies – Ada-Europe 2006. XII, 241 pages. 2006.

Vol. 4005: G. Lugosi, H.U. Simon (Eds.), Learning Theory. XI, 656 pages. 2006. (Sublibrary LNAI).

Vol. 4004: S. Vaudenay (Ed.), Advances in Cryptology - EUROCRYPT 2006. XIV, 613 pages. 2006.

Vol. 4003: Y. Koucheryavy, J. Harju, V.B. Iversen (Eds.), Next Generation Teletraffic and Wired/Wireless Advanced Networking. XVI, 582 pages. 2006.

Vol. 4001: E. Dubois, K. Pohl (Eds.), Advanced Information Systems Engineering. XVI, 560 pages. 2006.

Vol. 3999: C. Kop, G. Fliedl, H.C. Mayr, E. Métais (Eds.), Natural Language Processing and Information Systems. XIII, 227 pages. 2006.

Vol. 3998: T. Calamoneri, I. Finocchi, G.F. Italiano (Eds.), Algorithms and Complexity. XII, 394 pages. 2006.

Vol. 3997: W. Grieskamp, C. Weise (Eds.), Formal Approaches to Software Testing. XII, 219 pages. 2006.

Vol. 3996: A. Keller, J.-P. Martin-Flatin (Eds.), Self-Managed Networks, Systems, and Services. X, 185 pages. 2006.

Vol. 3995: G. Müller (Ed.), Emerging Trends in Information and Communication Security. XX, 524 pages. 2006.

Vol. 3994: V.N. Alexandrov, G.D. van Albada, P.M.A. Sloot, J. Dongarra, Computational Science – ICCS 2006, Part IV. XXXV, 1096 pages. 2006.

Vol. 3993: V.N. Alexandrov, G.D. van Albada, P.M.A. Sloot, J. Dongarra, Computational Science – ICCS 2006, Part III. XXXVI, 1136 pages. 2006.

Vol. 3992: V.N. Alexandrov, G.D. van Albada, P.M.A. Sloot, J. Dongarra, Computational Science – ICCS 2006, Part II. XXXV, 1122 pages. 2006.

Table of Contents

Creating Objects in the Flexible Authorization Framework*

Nicola Zannone[1,2], Sushil Jajodia[2], and Duminda Wijesekera[2]

[1] Dep. of Information and Communication Technology
University of Trento
zannone@dit.unitn.it
[2] Center for Secure Information Systems
George Mason University
{jajodia, dwijesek}@gmu.edu

Abstract. Access control is a crucial concern to build secure IT systems and, more specifically, to protect the confidentiality of information. However, access control is necessary, but not sufficient. Actually, IT systems can manipulate data to provide services to users. The results of a data processing may disclose information concerning the objects used in the data processing itself. Therefore, the control of information flow results fundamental to guarantee data protection. In the last years many information flow control models have been proposed. However, these frameworks mainly focus on the detection and prevention of improper information leaks and do not provide support for the dynamical creation of new objects.

In this paper we extend our previous work to automatically support the dynamical creation of objects by verifying the conditions under which objects can be created and automatically associating an access control policy to them. Moreover, our proposal includes mechanisms tailored to control the usage of information once it has been accessed.

1 Introduction

Access control is one of the main challenges in IT systems and has received significant attention in the last years. These efforts have matched with the development of many frameworks dealing with access control issues [1,2,3,4,5,6]. However, many of these proposals focus on the restriction on the release of information but not its propagation [7].

Actually, IT systems are developed not only to merely store data, but also to provide a number of functionalities designed to process data. Thereby, they may release information as part of their functionalities [8]. Yet, a malicious user can embed in some

* This material is based upon work supported by the National Science Foundation under grants IIS-0242237 and IIS-0430402. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. This work was partly supported by the projects RBNE0195K5 FIRB-ASTRO, 016004 IST-FP6-FET-IP-SENSORIA, 27587 IST-FP6-IP-SERENITY, 2003-S116-00018 PAT-MOSTRO.

E. Damiani and P. Liu (Eds.): Data and Applications Security 2006, LNCS 4127, pp. 1–14, 2006.

application provided by the IT system, a Trojan horse that, once the application is executed, copies sensitive information in a file accessible by the malicious user [9]. In this setting, information flow control plays a key role in ensuring that derived objects do not disclose sensitive information to unauthorized users.

This issue has spurred the research and development of frameworks that improve authorization frameworks with some form of flow control. Sammarati et al. [10] proposed to detect unauthorized information flow by checking if the set of authorizations associated with a derived object are a subset of the intersection of the sets of authorizations associated with the objects used to derive it. Similar approaches [11,12] have associated with each object an access control list that is propagated together with the information in the object. However, in these approaches the creation of objects is implicit. Essentially, they attempt to identify leaking information, but do not deal with the creation of new objects.

Moreover, this approach is to rigid to implement real access control policies. Actually, it is not flexible enough to support information declassification [8]. For instance, the US Privacy Act allows an agency to disclose information to those officers and employees of the agency who need it to perform their duties without the consent of the data subject. Furthermore, the Act does not impose any constraint to data that do not disclose personal identifying information.

In this paper, we extend our previous work [13] in order to automatically enforce access control policies on objects dynamically created in Flexible Authorization Framework (FAF) [14]. This requires to deal with some issues:

- deciding if an object can be created;
- associating authorizations with the new object;
- verifying if the derived object does not disclose sensitive information to unauthorized users.

The first issue is addressed by introducing conditions under which a data processing can be performed and enforcing the system to verify them before creating new objects. To cope with the second issue, we allow system administrators to define the policies governing access to derived objects, based on the authorizations associated with the objects used to derive them.

However, this is not sufficient to fully guarantee data protection. Actually, if a user is authorized to execute an application in which a Trojan horse is embedded, such a malicious application is considered as legitimate by the authorization framework. To this end, we propose an approach based on [10,11,12] to block non safe information flow. However, it is up to system administrators to decide whether or not an information flow is safe. Thereby, we only provide support for detecting flows of information that may be harmful to data subjects.

Other issues come up when the proposed approach is integrated in FAF. Actually, its current architecture does not support the dynamical creation of objects. To this intent, we need to improve it together with its underlying logic-based framework.

The remainder of the paper is structured as follows. Next (§2) we provide a brief overview of FAF. Then, we illustrate our approach for dealing with the dynamical creation of objects (§3) and for automatically deriving their access control policy (§4).

Next, we propose a mechanism to control information flow and show how such a mechanism copes with the Trojan horse problem (§5). Finally, we discuss related work (§6) and conclude the paper (§7).

2 Flexible Authorization Framework

Flexible Authorization Framework (FAF) [14] is a logic-based framework developed to manage access to data by users. It consists of four stages that are applied in sequence. The first stage takes in input the extensional description of the system, as subject and object hierarchies and a set of authorizations, and propagates authorizations through the organizational structure of the system. However, in this stage it is possible to derive contradictory authorizations, that is, a subject could be authorized and denied to execute an action on an object at the same time. The second stage aims to resolve this problem by applying conflict resolution policies. Once authorizations are propagated and conflicts resolved, there is the possibility that some access is neither authorized nor denied. In the third stage, decision policies are used to ensure the completeness of authorizations. In the last stage, specific domain properties are verified using integrity constraints, and all authorizations that violate them are removed.

FAF provides a logic-based language, called authorization specification language (**ASL**), tailored for encoding security needs. Before defining the language, we introduce the logic programming terminology needed to understand the framework. Let p be a predicate with arity n, and $t_1, \ldots, t_n$ be its appropriate terms. $p(t_1, \ldots, t_n)$ is called *atom*. Then, the term *literal* denotes an atom or its negation. **ASL** syntax includes the following predicates:

- A ternary predicate cando. Literal $\mathsf{cando}(o, s, a)$ is used to represent authorizations directly defined by the system administrator where o is an object, s is a subject, and a is a signed action terms. Depending on the sign, authorizations are permissions or prohibitions.
- A ternary predicate dercando that has the same arguments of predicate cando and is used to represent authorizations derived through propagation policies.
- A ternary predicate do that has the same arguments of predicate cando and represents effective permissions derived by applying conflicts resolution and decision policies.
- A 5-ary predicate done that is used to describe the actions executed by users. Intuitively, $\mathsf{done}(o, s, r, a, t)$ holds if subject s playing role r has executed action a on object o at time t.
- A propositional symbol error. Its occurrence in the model corresponds to a violation of some integrity constraints.
- A set of hie-predicates. In particular, the ternary predicate $\mathsf{in}(x, y, \mathsf{H})$ is used to denote that $x \leq y$ in hierarchy H.
- A set of rel-predicates. They are specific domain predicates.

Based on the architecture previously presented, every authorization specification **AS** is a locally stratified program where stratification is implemented by assigning levels to predicates (Table 1 [14]). For any specification **AS**, $\mathbf{AS}_i$ denotes the rules belonging to the i-th level.

Table 1. Strata in FAF specification

Stratum	Predicate	Rules defining predicate
$\mathbf{AS}_0$	hie-predicates	base relations.
	rel-predicates	base relations.
	done	base relation.
$\mathbf{AS}_1$	cando	the body may contain done, hie- and rel-literals.
$\mathbf{AS}_2$	dercando	the body may contain cando, dercando, done, hie- and rel-literals. Occurrences of dercando literals must be positive.
$\mathbf{AS}_3$	do	the head must be of the form $\mathsf{do}(\_, \_, +a)$ and the body may contain cando, dercando, done, hie- and rel-literals.
$\mathbf{AS}_4$	do	the head must be of the form $\mathsf{do}(o, s, -a)$ and the body contains the literal $\neg\mathsf{do}(o, s, +a)$.
$\mathbf{AS}_5$	error	the body may contain cando, dercando, do, done, hie- and rel-literals.

For optimizing the access control process, Jajodia et al. [14] proposed a materialized view architecture, where instances of predicates corresponding to views are maintained. Because predicates belong to strata, the materialization structure results (locally) stratified. This guarantees that the specification has a unique stable model and well-founded semantics [15,16]. Following [14], we use the notation $\mathcal{M}(\mathbf{AS})$ to refer to the unique stable model of specification **AS**.

3 Creating Objects

When a user requires to perform a data processing, the IT system should verify whether or not such a user has all necessary authorizations. In the remainder of this section, we address this issue.

Let O be the name space of all possible objects that may occur in the specification. We assume that they are organized into a hierarchical structure. This means that all possible objects are fully classified with respect to their type. Further, we assume that objects do not exist until they are created. This means that objects (together with their classification) may be not in the scope of the specification, although they are defined in O. Essentially, we assume that a possible object is considered only if some event demands its existence, that is, it is created.

Following [17], we introduce predicate exists, where $\mathsf{exists}(o)$ holds if object o exists, that is, it is already created. We define the *state of the system* as the set of existing objects and their relationships. To deal with the creation of objects, Liskov et al. [18] introduced two kinds of functions: *constructors* and *creators*. Constructors of a certain type return new objects of the corresponding type and creators initialize them. Essentially, constructors add object identifiers (i.e., names) to the state of the system and creators assign a value to such names. However, this approach distinguishes the identifier of an object from the values the object can assume. We merge this pair of functions into a single function, called *initiator*. Essentially, when an object is created, it exists together with its value. This allows us to be consistent with the semantics of FAF. Further, we assume that objects are never destroyed. From these assumptions, we

can deduce that the set of objects belonging to a state of the system is a subset of the set of objects belonging to the next state.

IT systems process data as part of their functionalities by providing automatic procedures used to organize and manipulate data. As done in [13], we represent data processing through initiators and make explicit the objects used by data processing and the users who performs them. Thus, we introduce an initiator for each procedure supported by the IT system. For instance, we write

$$f(s, o_1, \ldots, o_m) = o$$

to indicate that object o is the result of data processing f when this is performed by subject s and objects $o_1, \ldots, o_m$ are passed as input.[1] We assume that when an object is created (i.e., it enters in the scope of the specification), also its classification belongs to the specification. Notice that initiators do not belong to the specification language. We use them only to emphasize the objects used in the procedure and the subject that executes it.

Subjects may need to access exiting objects in order to create new objects. Moreover, only users that play a certain role or belong to a certain group may be entitled to perform a certain data processing. This means that an authorization framework should verify whether the subject has enough rights to access all objects needed to create the new one and whether he can execute the procedure.

Our idea is to enforce the system to verify the capabilities of the subject before an object is created. Based on this intuition, we redefine initiator f as

$$f(s, o_1, \ldots, o_m) = \begin{cases} o & \text{if } \mathcal{C} \text{ is true} \\ \perp & \text{otherwise} \end{cases}$$

where $\mathcal{C}$ represents the condition that must be satisfied and $\perp$ means that object o cannot be created since s does not have sufficient rights to execute the procedure.

Initiators are implemented in our framework through rules, called *creating rules*. These rules enforce the system to verify the conditions under which a user can create the object.

Definition 1. *Let f be an initiator, s be the subject executing f, $o_1, \ldots, o_m$ be the objects required by f, and $o = f(s, o_1, \ldots, o_m)$ be the derived object. A* creating rule *has the form*

$$\mathsf{exists}(o) \leftarrow L_1 \;\&\; \ldots \;\&\; L_n \;\&\; \mathsf{exists}(o_1) \;\&\; \ldots \;\&\; \mathsf{exists}(o_m).$$

where $L_1, \ldots, L_n$ are cando, dercando, do, done, hie-, *or* rel-*literals.* cando, dercando, do *literals may refer only to $o_1, \ldots, o_m$.*

Essentially, the conjunction of literals $L_1, \ldots, L_n$ represents the condition that a subject must satisfy in order to create object o. Last part of the body of the rule ensures that all objects necessary to create the new object already exist.

[1] Notice that initiators are not total functions since if one combines personal data of different users for creating an account, such account is not a valid object.

Example 1. A bank needs customer personal information, namely name, shipping address, and phone number, for creating an account. The bank IT system provides the procedure $openA$ for creating new accounts. Suppose a customer discloses his name (n), shipping address (sa), and phone number (p) to the bank. A bank employee s will be able to create $account$ ($= openA(s, n, sa, p)$) only if it is authorized to read customer data and he works in the Customer Services Division (CSD). In symbol,

$$\mathsf{exists}(account) \leftarrow \mathsf{do}(n, s, +\mathsf{read}) \;\&\; \mathsf{do}(sa, s, +\mathsf{read}) \;\&\; \mathsf{do}(p, s, +\mathsf{read}) \;\&\; \mathsf{in}(s, \mathit{CSD\text{-}employee}, \mathsf{ASH}) \;\&\; \mathsf{exists}(n) \;\&\; \mathsf{exists}(sa) \;\&\; \mathsf{exists}(p).$$

The outcome of a data processing may then be used to derive further objects. We represent the process to create an object as a tree, called *creation tree*, where the root is the "final" object and the leaves are *primitive objects* (i.e., objects that are directly stored in the system by users). In order to rebuild the creation tree, the system should keep trace of the process used to create the object. To this end, we introduce the binary predicate derivedFrom where $\mathsf{derivedFrom}(o_1, o_2)$ is used to indicate that object o_2 is used to derive object o_1. As for classification literals, derivedFrom literals referring an object are added to the model only when the object is created.

Example 2. Back to Example 1, the bank IT system stores the following set of literals:

$$\{\mathsf{derivedFrom}(account, n), \mathsf{derivedFrom}(account, sa), \mathsf{derivedFrom}(account, p)\}$$

4 Associating Authorizations with New Objects

Once an object has been created, authorizations should be associated with it. Since the object is not independent from the objects used to derive it, the policy associated with it should take into account the authorizations associated with them. Some proposals [11,12] associate with each object an access control list (ACL) that is propagated together with the information in the object. Essentially, the ACL associated with the new object is given by the intersection of all ACLs associated with the objects used to create it. However, when a system administrator specifies an access control policy for derived objects, he should consider that not all data processing disclose individually identifiable information [8]. For example, the sum of all account balances at a bank branch does not disclose data that allows to recover information associating a user with his own account balance.

We propose a flexible framework in order to allow system administrators to determine how authorizations are propagated to new objects. The idea is that authorizations associated with the objects used to derive the new one can be used to determine the authorizations associated with it. However, this approach cannot be directly implemented in FAF since the specification results no more stratified [13]. Next, we propose how FAF can be modified in order to support access control on derived objects maintaining the locally stratified structure.

4.1 Redefining Rules

To maintain the locally stratified structure, we need to redefine *creating rules*, *authorization rules* [14], *derivation rules* [14], and *positive decision rules* [14] by enforcing