Modelling Security Properties in a Grid-based Operating System with Anti-Goals

Alvaro Arenas Benjamin Aziz Juan Bicarregui Brian Matthews
Erica Y. Yang
STFC e-Science Centre, Rutherford Appleton Laboratory, Didcot, OX11 0QX, UK
{a.e.arenas, b.aziz, j.c.bicarregui, b.m.matthews, y.yang}@rl.ac.uk

Abstract

In this paper, we discuss the use of formal requirements-engineering techniques in capturing security requirements for a Grid-based operating system. We use KAOS goal model to represent two security goals for Grid systems, namely authorisation and single-sign on authentication. We apply goal-refinement to derive security requirements for these two security goals and we develop a model of anti-goals and show how system vulnerabilities and threats to the security goals can arise from such anti-models.

1 INTRODUCTION

Goal-driven approaches focus on why systems are constructed, providing the motivation and rationale for justifying software requirements. Examples of goal-oriented requirements methodologies include KAOS [11] and i^*/Tropos [2], among others. A goal is an objective that the system under development should achieve; this system includes the software as well as its environment.

This paper presents some fragments of the requirements engineering (RE) work for XtreemOS [8], a grid-based extension of the Linux operating system. We follow the KAOS approach [11], a goal-oriented RE methodology that includes activities such as the identification of the goals to be achieved by the envisioned system, the refinement of such goals and their operationalisation into specifications of services and constraints, and the assignment of responsibilities for the resulting requirements to agents such as humans, devices and software.

We describe the modelling of two key properties for XtreemOS, namely authorisation and single-sign on authentication, concentrating only on the goal and anti-goal models due to space limitation. We use goal-refinement techniques to derive security requirements for each of these properties. The anti-goal models are built to identify potential threats and vulnerabilities to the security properties.

The structure of the paper is as follows. Section 2 introduces the main characteristics of XtreemOS and Section 3 briefly describes the KAOS methodology. The core of the paper are Sections 4 and 5, where the goal and anti-goal models of the authorisation and single-sign on authentication requirements in XtreemOS are defined. Section 6 relates our work with other approaches. Finally, Section 7 concludes the paper and highlights future work.

2 XtreemOS: A Grid-based OS

XtreemOS [1] is a European project which has the objective to develop an open source Grid operating system (named XtreemOS) which supports scientific and commercial Grid applications. The goal is to provide an abstract interface to its underlying distributed physical resources, as a traditional operating system does for a single computer. The approach being investigated is to base XtreemOS on the existing Linux operating system. A set of system services, extending those found in the traditional Linux, will provide users with fully integrated Grid capabilities.

A key feature of XtreemOS is native support for Virtual Organisations (VOs), where a VO can be seen as a temporary or permanent coalition of geographically dispersed entities (individuals, groups, organisational units or entire organisations) that pool resources, capabilities and information to achieve common objectives.

XtreemOS consists of a wide range of Grid services (e.g. integrated application execution and data management) running in use space of Linux and making extensive use of Linux kernel modules (e.g. Pluggable Authentication Module - PAM and Filesystem in Userspace - FUSE) to deliver the Grid capabilities. Although, in general terms, many security requirements (e.g. authorisation and authentication) have been well studied in the Grid security community, it

[1] http://www.xtreemos.eu/

remains to be shown how these conventional security requirements can be met in this new setting.

The two selective security properties (i.e. authorisation and single-sign on authentication) are fundamental to Grid systems, and are central to the majority of our applications. They have been analysed informally in the XtreemOS security requirement study [14]. The work described in this paper, arises from the need to thoroughly and rigorously understand the security requirements of the critical components of XtreemOS and of the critical interactions among system components. By performing this task, we aim to highlight and ultimately eliminate the presence of well-known security vulnerabilities from our design and implementation. We also expect to provide feedback to system designers along the security design process and use the outcome from this analysis to create test cases for XtreemOS.

3 The KAOS Methodology

KAOS is a generic methodology that is based on the capture, structuring and precise formulation of the system goals [11]. A goal is prescriptive description of system properties, formulated in non-operational terms. A system includes not only the software to be developed but also its environment. Goals are refined and operationalised in a top-down manner as the system is designed or bottom up approach while re-engineering existing systems. The approach also supports adverse environments, composed of possibly malicious external agents trying to undermine the system goal rather than to collaborate in the goal fulfillment. As a Grid system is typically composed of a large number of nodes interacting in an open and possibly adverse environment, this approach fits our needs well.

A KAOS model is composed of a number of sub-models. The **goal model** captures and structures the assumed and required properties of a system by formalising a property as a top-level goal which is then refined to intermediate subgoals and finally to low-level requirements, which represent system goals that can be made operational. The **agent model** assigns goals to agents in a realizable way. Discovering the responsible agents is the criterion to stop a goal-refinement process. The **object model** is used to identify the concepts of the application domain that are relevant with respect to the requirements and to provide static constraints on the operational systems that will satisfy the requirements. The object model consists of objects from the stakeholders' domain and objects introduced to express requirements or constraints on the operational system. The **operation model** details, at state-transition level, the actions an agent has to perform to reach the goals it is responsible for. Finally, the **anti-goal model** captures attacks on the system and how they are addressed. This model is built in parallel with the goal-model and helps discover goals that will improve the robustness of the system, especially against malicious agents whose aim is to break the goals of the system.

In what follows, we shall concentrate on the use of the goal model to represent security properties and the use of the anti-goal model to express certain attacks and vulnerabilities undermining these properties. Both goals and anti-goals are formally modeled using Linear Temporal Logic (LTL) formulae. In our definitions, we only require a subset of LTL represented by the $\Box P$ and $\Diamond P$ operators, where P is defined in terms of the logic connectors $(\wedge \vee \neg \rightarrow \leftrightarrow)$. The former states that P has to hold from this point in time and in all subsequent states, whereas the latter states that P has to hold at some time in the future. We also write $(P \Rightarrow Q)$ to mean $\Box(P \rightarrow Q)$ and $(P \Leftrightarrow Q)$ to mean $(P \Rightarrow Q) \wedge (P \Leftarrow Q)$.

Goals may be organized in AND/OR refinement-abstraction hierarchies, where higher-level goals are in general strategic, coarse-grained and involve multiple agents whereas lower-level goals are in general technical, fine-grained and involve fewer agents. In such structures, AND-refinement links relate a goal to a set of sub-goals possibly conjoined with domain properties or environment assumptions; this means that satisfying all subgoals in the refinement is a sufficient condition in the domain for satisfying the goal. OR-refinement links may relate a goal to a set of alternative refinements. Goal refinement ends when every sub-goal is realizable by some individual agent assigned to it. A requirement is a terminal goal under responsibility of an agent in the software-to-be. An expectation is a terminal goal under responsibility of an agent in the environment.

In AND-refinement hierarchies, one needs to show that completeness and minimality are preserved. Assume that a goal G is refined to goals $G_1 \ldots G_n$, then completeness is defined as $G_1 \wedge \ldots \wedge G_n \Rightarrow G$ whereas minimality is defined as $G \Rightarrow G_1 \wedge \ldots \wedge G_n$. Completeness states that when all of $G_1 \ldots G_n$ subgoals are achieved, then G is achieved. Hence a goal is *completed* by the completion of its subgoals. By contrast, minimality is concerned with the *minimum* number of subgoals implied in the achievement of a higher-level goal. Thus, if any of the subgoals is not achieved, the goal will not be achieved.

In the anti-goal model [12], the strategy consists in breaking a security goal by negating its top-level definition and then expanding the negation by substituting domain-specific definitions to specific vulnerabilities.

4 Authorisation

In general, authorisation in an operating system is used to protect the computing resources underlying the operating system by permitting access to those resource to certain users with access rights. In a grid-based operating system,

such as XtreemOS, the underlying resources could be grid-based as well as local.

The requirement to have authorisation in XtreemOS was based on the requirements to have data storage confidentiality and integrity. These two requirements appeared as R78 and R80, repsectivley, in the security requirements for the XtreemOS application scenarios document [14]. The two requirements identify *access control* as the main mechanism for ensuring authorisation. Therefore, here we do not consider other definitions of confidentiality and integrity, such as the ones based on information flow or non-interference properties.

Based on the definitions of R78 and R80, only valid *principals* of the Grid are authorised to access *data* stored on resources. *Principals* are defined as being the users, administrators or services present in the Grid, whereas *data* is defined as being any data stored on the local or grid-based filesystem, data present in a shared memory or data contained within a license. A valid principal is defined as being either one of the following: *The owner of the data, or a principal who is a member of a VO and has the access right issued by the VO as well as being assigned to a task requiring access to the data.*

The last condition assumes the principle of *least privilege*, which states that principals should obtain no more than their minimum (access) rights necessary to achieve their functionality (task). The right issued by the VO can be either "read" or "write". This will determine whether the security property modelled is confidentiality or integrity.

4.1 A Goal Model for Authorisation

The goal model of the authorisation property is illustrated informally in Figure 1, where circles represent AND-relations. Before defining the formal goal model for authorisation, we introduce a few useful sets and predicates:

- *VO*: the set of all VOs.
- *Data*: the set of all data including that which may be created and used by XtreemOS applications.
- *Principal*: the set of all principals. Principals may be users, administrators or Grid services.
- *Task*: the set of tasks running in a VO. Tasks are usually assigned to principals in the VO.
- *Entity*: is the union set $VO \cup Data \cup Task$.
- *Credential*: The set of all credentials. A credential may be a password, a certificate or any other mechanism.
- *Attribute*: The set of all attributes of a credential. This set includes data ownership, VO membership, task assignment and data read and write attributes.
- *Right*: The set of *read* and *write* access rights. Note that $Right \subset Attribute$.
- $owner : Principal \times Data \rightarrow \mathbb{B}$ is a predicate denoting that a principal owns a dataset.
- $member : Principal \times VO \rightarrow \mathbb{B}$ is a predicate denoting that a principal is a member of a VO.
- $assigned : Principal \times Task \rightarrow \mathbb{B}$ is a predicate denoting that a principal is assigned to some task.
- $requires : Task \times Data \rightarrow \mathbb{B}$ is a predicate denoting that a task requires a dataset.
- $hasVORights : Principal \times Data \times VO \times Right \rightarrow \mathbb{B}$ is a predicate on principals and datasets denoting that a principal has a right (read or write) to access the dataset and that the right was issued by some VO.
- $binding : Principal \times Entity \times Attribute \times Credential \rightarrow \mathbb{B}$ is a predicate on principals and entities denoting that a principal is bound to the entity by means of some credential and this binding has the specified attribute.
- $issued\_By : Credential \times VO \rightarrow \mathbb{B}$ is a predicate stating that a credential is issued by a VO.

Using these sets and predicates, we can define the formal top-level goal of authorisation as follows:

Goal [Authorisation]
FormalDef $\forall p \in Principal, d \in Data, r \in Right :$
$authorised(p, d, r) \Leftrightarrow$
$(owner(p, d) \vee (\exists vo \in VO, t \in Task : member(p, vo) \wedge hasVORights(p, d, vo, r) \wedge assigned(p, t) \wedge requires(t, d)))$

where $authorised : Principal \times Data \times Right \rightarrow \mathbb{B}$ is a predicate denoting that a principal is authorised to access some data with some right (read or write).

Next, we break down the top-level goal in terms of the following two subgoals:

Goal [Authorisation Safety]
FormalDef $\forall p \in Principal, d \in Data, r \in Right :$
$authorised(p, d, r) \Rightarrow$
$(owner(p, d) \vee (\exists vo \in VO, t \in Task : member(p, vo) \wedge hasVORights(p, d, vo, r) \wedge assigned(p, t) \wedge requires(t, d)))$

Goal [Authorisation Liveness]
FormalDef $\forall p \in Principal, d \in Data, r \in Right :$
$authorised(p, d, r) \Leftarrow$
$(owner(p, d) \vee (\exists vo \in VO, t \in Task : member(p, vo) \wedge hasVORights(p, d, vo, r) \wedge assigned(p, t) \wedge requires(t, d)))$

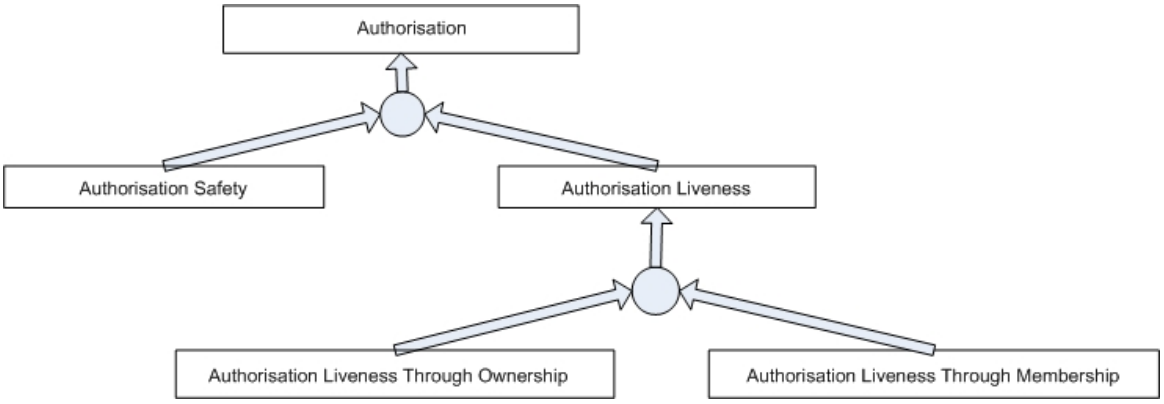

Figure 1. The Goal Model for Authorisation.

The first subgoal can only be true if it is true that the principal is either the owner or a valid member of a VO. Therefore, it denotes safety of authorisation through the sufficiency of the right side condition. In the second subgoal, the left side must be true if the principal is the owner of the data or a valid VO member for the goal to be true. Therefore, it denotes liveness of authorisation. It is straightforward to show that both the above two subgoals are complete and minimal with respect to the main authorisation goal.

Now, we break down the second subgoal further on to the following two subgoals:

Goal [Authorisation Liveness Through Ownership]
FormalDef $\forall p \in \mathit{Principal}, d \in \mathit{Data}, r \in \mathit{Right}$:
$\mathit{authorised}(p, d, r) \Leftarrow \mathit{owner}(p, d)$

Goal [Authorisation Liveness Through Membership]
FormalDef $\forall p \in \mathit{Principal}, d \in \mathit{Data}, r \in \mathit{Right}$:
$\mathit{authorised}(p, d, r) \Leftarrow (\exists vo \in \mathit{VO}, t \in \mathit{Task} : \mathit{member}(p, vo) \wedge \mathit{hasVORights}(p, d, vo, r) \wedge \mathit{assigned}(p, t) \wedge \mathit{requires}(t, d))$

Note that r in the first subgoal does not affect the predicate since an owner has exclusive rights over its datasets. We now give Grid-based domain-specific definitions of the predicates appearing on the right side of these implications.

Definition [Owner Credential Validated]
FormalDef $(\forall p \in \mathit{Principal}, d \in \mathit{Data}) : \mathit{owner}(p, d) \Leftrightarrow (\exists cr \in \mathit{Credential} : \mathit{binding}(p, d, \mathit{own}, cr) \wedge \mathit{well\_defined}(cr))$

Definition [Assignment Credential Validated]
FormalDef $(\forall p \in \mathit{Principal}, t \in \mathit{Task}) : \mathit{assigned}(p, t) \Leftrightarrow (\exists cr \in \mathit{Credential} : \mathit{binding}(p, t, \mathit{assigned}, cr))$

Definition [Member Credential Validated]
FormalDef $(\forall p \in \mathit{Principal}, vo \in \mathit{VO}) : \mathit{member}(p, vo) \Leftrightarrow (\exists cr \in \mathit{Credential} : \mathit{binding}(p, vo, \mathit{member}, cr))$

Definition [Rights Credential Validated]
FormalDef $(\forall p \in \mathit{Principal}, d \in \mathit{Data}, vo \in \mathit{VO}, r \in \mathit{Right}) : \mathit{hasVORights}(p, d, vo, r) \Leftrightarrow (\exists cr \in \mathit{Credential} : \mathit{binding}(p, d, r, cr) \wedge \mathit{issued\_By}(cr, vo))$

The last definition may be specialised into the following two instances using read and write credentials.

Definition [Read Credential Validated]
FormalDef $(\forall p \in \mathit{Principal}, d \in \mathit{Data}, vo \in \mathit{VO}) : \mathit{hasVORights}(p, d, vo, \mathit{read}) \Leftrightarrow (\exists cr \in \mathit{Credential} : \mathit{binding}(p, d, \mathit{read}, cr) \wedge \mathit{issued\_By}(cr, vo))$

Definition [Write Credential Validated]
FormalDef $(\forall p \in \mathit{Principal}, d \in \mathit{Data}, vo \in \mathit{VO}) : \mathit{hasVORights}(p, d, vo, \mathit{write}) \Leftrightarrow (\exists cr \in \mathit{Credential} : \mathit{binding}(p, d, \mathit{write}, cr) \wedge \mathit{issued\_By}(cr, vo))$

In addition to the above definitions, we state the well-definedness of ownership credentials as follows:

Definition [Credential Well-Definedness]
FormalDef $\forall cr \in \mathit{Credential}$:
$\mathit{well\_defined}(cr) \Leftrightarrow (\forall p, p' \in \mathit{Principal}, d \in \mathit{Data} : \mathit{binding}(p, d, \mathit{own}, cr) \wedge \mathit{binding}(p', d, \mathit{own}, cr) \Rightarrow p = p')$

The requirement essentially states that a credential cannot refer to more than one principal. An example of this requirement is that a certificate, which has a unique serial number, must refer to only one principal. The public key of the principal contained in the certificate represents the data to which the principal is bound.

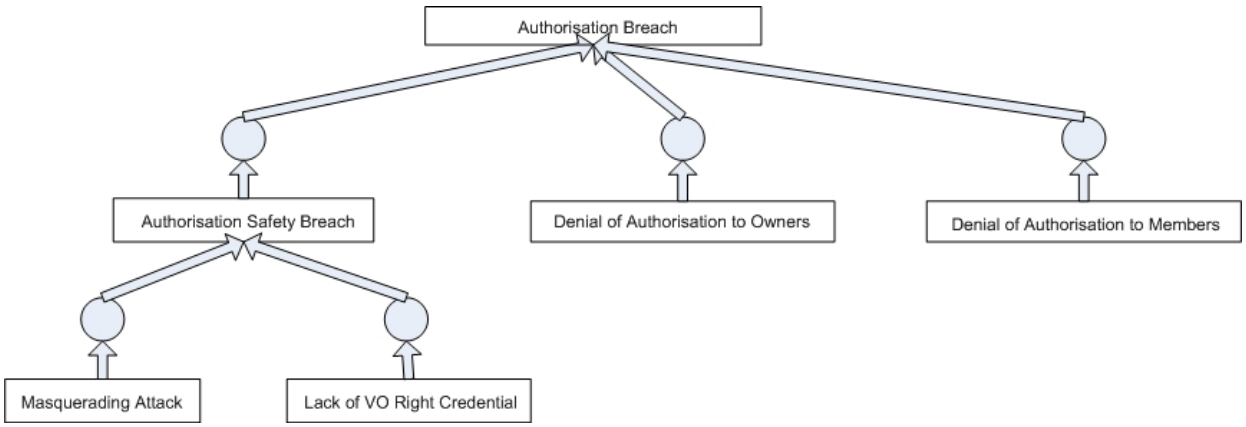

Figure 2. The Anti-Goal Model for Authorisation.

4.2 An Authorisation Anti-Goal Model

Our strategy in deriving system vulnerabilities is based on the idea of deriving anti-goals from their corresponding goal definitions by negating those definitions. In fact, it is sufficient to negate low-level subgoals, since by minimality, this will ensure that the top-level goal is negated and hence the property is breached. After that, domain-specific definitions are used to expand the anti-goals in order to obtain domain-specific vulnerabilities. The anti-goal model for the breach of authorisation is shown in Figure 2, where multiple arrows leading to the same goal represent OR-relations.

To obtain the formal model, we start by negating the "Authorisation Safety", "Authorisation Liveness Through Ownership" and "Authorisation Liveness Through Membership" subgoals, as follows:

AntiGoal [Authorisation Safety Breach]
FormalDef $\exists p \in \mathit{Principal}, d \in \mathit{Data}, r \in \mathit{Right} : \Diamond(\mathit{authorised}(p,d,r) \wedge \neg(\mathit{owner}(p,d) \vee (\exists vo \in \mathit{VO}, t \in \mathit{Task} : \mathit{member}(p,vo) \wedge \mathit{hasVORights}(p,d,vo,r) \wedge \mathit{assigned}(p,t) \wedge \mathit{requires}(t,d))))$

AntiGoal [Denial of Authorisation to Owners]
FormalDef $\exists p \in \mathit{Principal}, d \in \mathit{Data}, r \in \mathit{Right} : \Diamond(\neg\mathit{authorised}(p,d,r) \wedge \mathit{owner}(p,d))$

AntiGoal [Denial of Authorisation to Members]
FormalDef $\exists p \in \mathit{Principal}, d \in \mathit{Data}, r \in \mathit{Right} : \Diamond(\neg\mathit{authorised}(p,d,r) \wedge (\exists vo \in \mathit{VO}, t \in \mathit{Task} : \mathit{member}(p,vo) \wedge \mathit{hasVORights}(p,d,vo,r) \wedge \mathit{assigned}(p,t) \wedge \mathit{requires}(t,d)))$

Though interesting, the last two anti-subgoals are outside the scope of confidentiality and integrity breaches since these deal more with the denial of service-like vulnerabilities. Therefore, we only focus on the first anti-subgoal. It is possible to expand the "Authorisation Safety Breach" anti-subgoal by substituting the definitions of the different predicates into the anti-subgoal. However, for lack of space, we shall consider only the interesting cases of the ownership and the VO rights predicates. We start with the former:

AntiGoal [Authorisation Safety Breach]
FormalDef $\exists p \in \mathit{Principal}, d \in \mathit{Data}, r \in \mathit{Right}, \forall cr \in \mathit{Credential}, vo \in \mathit{VO}, t \in \mathit{Task} : \Diamond(\mathit{authorised}(p,d,r) \wedge (\neg\mathit{binding}(p,d,\mathit{own},cr) \vee \neg\mathit{well\_defined}(cr)) \wedge (\neg\mathit{member}(p,vo) \vee \neg\mathit{hasVORights}(p,d,vo,r) \vee \neg\mathit{assigned}(p,t) \vee \neg\mathit{requires}(t,d)))$

This anti-subgoal introduces the predicate of badly defined certificates, $\neg\mathit{well\_defined}(cr)$. By substituting the definition of this predicate into the anti-subgoal, we get.

AntiGoal [Masquerading Attack]
FormalDef $\exists p, p' \in \mathit{Principal}, d \in \mathit{Data}, r \in \mathit{Right}, \forall cr \in \mathit{Credential}, vo \in \mathit{VO}, t \in \mathit{Task} : \Diamond(\mathit{authorised}(p,d,r) \wedge (\neg\mathit{binding}(p,d,\mathit{own},cr) \vee (\mathit{binding}(p,d,\mathit{own},cr) \wedge \mathit{binding}(p',d,\mathit{own},cr) \wedge \neg(p=p'))) \wedge (\neg\mathit{member}(p,vo) \vee \neg\mathit{hasVORights}(p,d,vo,r) \vee \neg\mathit{assigned}(p,t) \vee \neg\mathit{requires}(t,d)))$

This anti-subgoal models the case in which a credential refers to two different principals both as owners of the same dataset. In our opinion, this constitutes a form of masquerading attacks in which one principal pretends to be another by presenting information to the system pertaining to be the other principal. Expanding with the definition of the lack of VO rights yields the following anti-subgoal:

AntiGoal [Lack of VO Right Credential]
FormalDef $\exists p \in \mathit{Principal}, d \in \mathit{Data}, r \in \mathit{Right}, \forall cr \in \mathit{Credential}, vo \in \mathit{VO}, t \in \mathit{Task} : \Diamond(\mathit{authorised}(p,d,r) \wedge (\neg\mathit{owner}(p,d)) \wedge (\neg\mathit{member}(p,vo) \vee (\neg\mathit{binding}(p,d,r,cr) \vee$

$\neg issued\_By(cr, vo)) \ \vee \ \neg assigned(p, t) \ \vee \ \neg requires(t, d)))$

This anti-subgoal expresses a hacking attack in which the principal is authorised to access the data even though it has no VO rights credential. More specifically, it is possible to get the following two instances of the anti-subgoal for the cases of confidentiality and integrity breaches:

AntiGoal [Confidentiality Breach]
FormalDef $\exists p \in Principal, d \in Data, r \in Right, \forall cr \in Credential, vo \in VO, t \in Task : \ \Diamond (authorised(p, d, read) \ \wedge \ (\neg owner(p, d)) \wedge (\neg member(p, vo) \ \vee \ (\neg binding(p, d, read, cr) \vee \neg issued\_By(cr, vo)) \ \vee \ \neg assigned(p, t) \ \vee \ \neg requires(t, d)))$

AntiGoal [Integrity Breach]
FormalDef $\exists p \in Principal, d \in Data, r \in Right, \forall cr \in Credential, vo \in VO, t \in Task : \ \Diamond (authorised(p, d, write) \ \wedge \ (\neg owner(p, d)) \wedge (\neg member(p, vo) \ \vee \ (\neg binding(p, d, write, cr) \vee \neg issued\_By(cr, vo)) \ \vee \ \neg assigned(p, t) \ \vee \ \neg requires(t, d)))$

4.3 Discussion on the Authorisation Model

Essentially, the authorisation goal model captures both the data storage confidentiality and integrity properties by generalising the access rights to those data. The distinction between read/write access rights is only significant in the case of VO members who do not qualify as owners of the data; owners are automatically assumed to have both rights. The model also captures liveness properties in the sense that authorisation must be granted if ownership or valid membership are established. The domain-specific definitions of the various ownership and valid membership predicates allow us to talk about concrete implementations of these predicates, which may introduce more predicates such as the well-definedness of credentials.

In the anti-goal model, we select a subset of the anti-goals generated by the negation of the authorisation goal, which represent breaches in confidentiality and integrity. The use of domain-specific definitions again in the anti-goal model reveals a few possible vulnerabilities that may arise later in the implementation phase, such as the masquerading attack and the lack of VO rights vulnerability.

5 Single-Sign On

Single-Sign On (SSO) [5] is a method of access control that allows the user to authenticate once and gain access to multiple resources without the need to re-authenticate again. SSO associates a unique identifier with every user in a VO or across VOs at authentication time such that changes made in VO resource configurations are transparent to the user and such that VO resources have a common identification mechanisms for all users. The requirement to have the SSO method of access control in XtreemOS was identified in D3.5.2 [14] as R82. The requirement identifies two main goals required by SSO: users must be authenticated and users have unique identities.

5.1 A Goal Model for SSO

The SSO goal model is illustrated informally in Figure 3. Next, we introduce a few useful predicates.

- *User*: The set of Grid users, where $User \subseteq Principal$.
- *ID*: The set of user ids.
- *authenticated* : $User \times VO \rightarrow \mathbb{B}$ is a predicate to state whether a user has been authenticated or not by a VO.
- *single_id* : $User \times VO \rightarrow \mathbb{B}$ is a predicate to state whether a user has a single id or not within a VO.

The formal model is then defined as the following goal, which introduces the *sso* : $User \times VO \rightarrow \mathbb{B}$ predicate to denote whether a user enjoys SSO within a VO or not:

Goal [Single-Sign On]
FormalDef $\forall u \in User, vo \in VO : \ sso(u, vo) \ \Rightarrow \ (authenticated(u, vo) \wedge single\_id(u, vo))$

The user authentication and single user id predicates are defined as follows.

Definition [User Authenticated]
FormalDef $\forall u \in User, vo \in VO : \ authenticated(u, vo) \ \Leftrightarrow \ (\exists cr \in Credential : \ binding(u, id, owns, cr) \ \wedge \ issued\_By(cr, vo))$

Definition [User Has Unique ID]
FormalDef $\forall u \in User, vo \in VO, cr, cr' \in Credential, id, id' \in ID : \ single\_id(u, vo) \ \Leftrightarrow \ (binding(u, id, owns, cr) \ \wedge \ binding(u, id', owns, cr') \wedge \ issued\_By(cr, vo) \ \wedge \ issued\_By(cr', vo) \ \Rightarrow \ id = id')$

In the first definition, a user is considered to be authenticated in a VO if and only if there is a credential binding the user to its identity (such as a certificate) and the credential is issued by the VO. The second definition states that a user must not have two different identities within a VO, even though it may have multiple (different) certificates issued by that VO.

5.2 An Anti-Goal Model for SSO

Our anti-goal model is illustrated in Figure 4. As in the case of authorisation, we define the formal anti-goal model of SSO by negating the top-level goal:

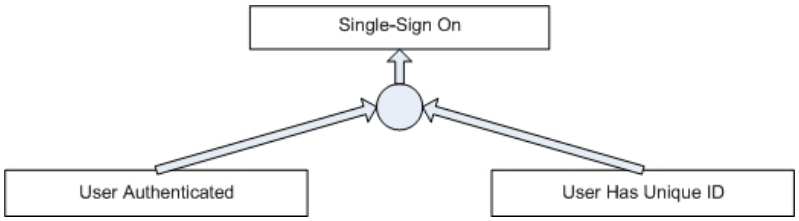

Figure 3. The Goal Model for SSO.

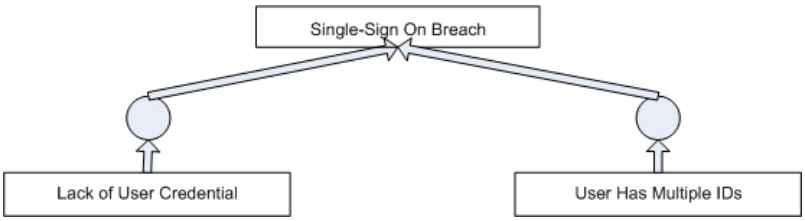

Figure 4. The Anti-goal Model for SSO.

AntiGoal [Single-Sign On Breach]
FormalDef $\Diamond\ (\exists u \in User, vo \in VO : sso(u, vo) \wedge (\neg authenticated(u, vo) \vee \neg single\_id(u, vo))$

Expanding the anti-subgoal along the definitions of the authentication and single user id predicates, we obtain the following two vulnerabilities:

AntiGoal [Lack of User Credential]
FormalDef $\Diamond\ (\exists u \in User, vo \in VO : sso(u, vo) \wedge ((\forall cr \in Credential : \neg binding(u, id, owns, cr) \vee \neg issued\_By(cr, vo)) \vee \neg single\_id(u, vo))$

AntiGoal [User has Multiple IDs]
FormalDef $\Diamond\ (\exists u \in User, vo \in VO : sso(u, vo) \wedge (\neg authenticated(u, vo) \vee (binding(u, id, owns, cr) \wedge binding(u, id', owns, cr') \wedge issued\_By(cr, vo) \wedge issued\_By(cr', vo) \wedge \neg(id = id')))$

The former represents the case of a user who is capable of obtaining the SSO capability but that does not have a valid identity credential issued by the VO. The second however refers to the case of the user who has managed to obtain multiple identities but it is still considered to have the SSO capability. This latter case is dangerous as it may lead an SSO-based system to assume that there are two users, and hence leads to a form of non-existent user attack.

5.3 Discussion on the SSO Models

The SSO goal model introduced a domain-specific definition of what it means for the user to be authenticated. There are other definitions, as such for example the running of a session of an authentication protocol between the user and the VO, with the minimum requirement that at the end of the protocol, the VO authenticates the user. However, such definitions have more complex formalisations that require past temporal operators. In the original requirements documents [14], SSO also requires re-authentication with the change of identities. We have not included this requirement here since it can be expressed as a re-application of the SSO process.

6 Related Work

There are several frameworks for modelling and analysing security requirements, such as Misuse Cases [10], Secure Tropos [4] and UMLsec [6]. Here, we have followed the Anti-Goals approach proposed in [12]. However, there are some differences in the way we have applied the anti-goals methodology. The modelling of security properties in [12] is based on real-time linear temporal logic extended with epistemic operators. Our modelling of security properties is based on pure linear temporal logic only. This has been sufficient to capture the properties we are interested in and it has allowed us to exploit current tool support for the methodology [9]. Further, we have not followed the confidentiality pattern suggested by [12], which is based on epistemic operators, but rather adopted a definition based on the use of credentials.

The work reported here is close in spirit to [7], which presents a preliminary threat analysis for XtreemOS. It follows the traditional attack-tree methodology: first, it identifies the main assets within XtreemOS that need protection;

then, it details potential attacks using attacker trees. The work was informal and was observed the need for a more rigorous approach, which has motivated to this work

There is a fresh interest in high-level security patterns for large-scale distributed systems such as Grids [1, 3]. In [1], it is presented a KAOS pattern for dynamic access control in Grids –the so-called Chinese Wall policy–. The pattern is also formalised using linear temporal logic, but it includes operators for referring to past time that are used for modelling history-based access control. In [3], it is presented a pattern for secure VO management in Grids. Our work could be seen as a previous step to the application of such a pattern, where the full set of KAOS models correspond to an abstract architecture that could be instatiated with such a VO management pattern.

7 Conclusion and Future Work

This paper has presented a goal-oriented modelling of authorisation and single-sign on properties in the XtreemOS Grid-based operating system. Our approach has consisted in applying the KAOS goal-oriented methodology, which formalises goal specification using linear temporal logic.

The use of anti-goals has been central for reasoning about security goals. They are used in KAOS as a way to guide the designers to think about security threats, in a similar way as engineers elicit safety hazards. With the application of anti-goals, we have discovered potential vulnerabilities such as bad management of certificates, and masquerading attack. These results, in effect, gave insight into the original security requirements captured in XtreemOS and helped strengthen those requirements.

As future work, we plan to analyse the usefulness of our single-sign on model in other Grid systems in order to determine if it could be defined as a pattern for modelling this important property. Some open issues remains on the application of KAOS for security. Although there is notion of completeness for handling obstacles [13], hence anti-goals, it is desirable to have a notion of completeness covering all potential threats to the system. We are also working toward developing a technique to formally derive test-cases from the requirements model, which we will apply to the security critical components of the XtreemOS architecture.

Acknowledgment

The authors would like to thank Philippe Massonet and Christophe Ponsard from CETIC for comments on early drafts of this paper. We are also grateful to the GridTrust project for facilitating the tool to model the security requirements. The work reported here was partially funded by the European Commission under grant FP6-033576 to the XtreemOS project.

References

[1] G. Dallons, P. Massonet, J. F. Molderez, C. Ponsard, and A. E. Arenas. An Analysis of the Chinese Wall Pattern for Guaranteeing Confidentiality in Grid-Based Virtual Organisations. In *International Workshop on Security, Trust and Privacy in Grid Systems, Grid-STP 2007*. IEEE, 2007.

[2] A. Fuxman, L. Liu, J. Mylopoulos, M. Pistore, M. Roveri, and P. Traverso. Specifying and Analyzing Early Requirements in Tropos. *Requirements Eng.*, 9(2):132–150, 2004.

[3] A. Gaeta, M. Gaeta, I. Djordjevic, A. Smith, T. Dimitrakos, M. Colombo, and S. Miranda. Design Patterns for Secure Virtual Organization Management Architecture. In *International Workshop on Security, Trust and Privacy in Grid Systems, Grid-STP 2007*. IEEE, 2007.

[4] P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone. Requirements Engineering for Trust Management: Model, Methodology, and Reasoning. *Journal of Information Security*, 5(4):257–274, 2006.

[5] J. Jensen, D. Spence, and M. Viljoen. Grid Single Sign-On in CCLRC. In *Proceedings of the UK e-Science All Hands Meeting 2006 (AHM2006)*, Nottingham, UK, Sept. 2006.

[6] J. Jurjens. *Secure Systems Development with UML*. Springer-Verlag, 2004.

[7] A. D. Lakhani, E. Yang, B. Matthews, I. Johnson, S. Naqvi, and G. C. Silaghi. Threat Analysis and Attacks on XtreemOS: A Grid-Enabled Operating System. In *Toward Next Generation Grids, Proceedings of the CoreGRID Symposium 2007*. Springer, 2007.

[8] C. Morin. XtreemOS: A Grid Operating Sytem Making your Computer Ready for Participating in Virtual Organizations. In *10th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC 2007)*. IEEE, 2007.

[9] C. Ponsard, P. Massonet, J. F. Molderez, A. Rifaut, A. van Lamsweerde, and H. T. Van. Early Verification and Validation of Mission Critical Systems. *Journal of Formal Methods in System Design*, 30(3), 2007.

[10] G. Sindre and A. L. Opdahl. Eliciting Security Requirements with Misuse Cases. *Requirements Engineering*, 10(1):34–44, 2005.

[11] A. van Lamsweerde. Requirements Engineering in the Year 00: A Research Perspective. In *International Conference on Software Engineering*, pages 5–19, 2000.

[12] A. van Lamsweerde. Elaborating Security Requirements by Construction of Intentional Anti-Models. In *26th ACM-IEEE International Conference on Software Engineering (ICSE'04)*, pages 148–157. IEEE Press, 2004.

[13] A. van Lamsweerde and E. Letier. Handling Obstacles in Goal-Oriented Requirements Engineering. *IEEE Transactions on Software Engineering*, 26(10):978–1005, 2000.

[14] XtreemOS. XtreemOS Deliverable D3.5.2: Security Requirements for a Grid-based OS, 2007.