A Formal Security Requirements Model for a Grid-based Operating System

Benjamin Aziz, Alvaro Arenas, Juan Bicarregui, Brian Matthews, Erica Yang
STFC e-Science Centre, Rutherford Appleton Laboratory
Didcot, OX11 0QX, UK
www.e-science.stfc.ac.uk/
{*b.aziz,a.e.arenas,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. Our approach is based on the KAOS methodology in which system goals can be refined to sets of requirements that can be satisfied by agents performing specific operations on system objects. We focus on the example of one security goal of interest to Grid-based systems, namely the authorisation to access data, and show how this goal can be refined into system requirements. Then we develop a model of anti-goals, and show how the model captures vulnerabilities that undermine the main security goal.

Keywords: Grid, Operating Systems, Requirement Engineering, Security

1. INTRODUCTION

We investigate in this paper the autherisation requirement for the XtreemOS project. XtreemOS [1] is a European project which has the objective to develop an open source Grid operating system, 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, authentication, data confidentiality, integrity etc.) have been well studied in the Grid security community, it remains to be shown how these conventional security requirements can be met in this new setting.

In this paper, we focus on one security property; namely that of authorisation, which is fundamental to Grid systems and is central to the majority of our applications. This property has been analysed informally in the XtreemOS security requirement study [5]. The work described here re-captures the requirement in a formal set-up using the KAOS requirement engineering methodology [2]. Such work 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 security vulnerabilities from our design and implementation. We also expect to

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

provide feedback to system designers along the security design process and use the outcome from this work to create test cases for the XtreemOS system.

The rest of the paper is structured as follows. In Section 2, we briefly describe the KAOS RE methodology. The core of the paper is Section 3, in which the goal and anti-goal models of autherisation in XtreemOS using access control are defined and discussed. Finally, Section 4 concludes the paper and highlights future work.

2. THE KAOS METHODOLOGY

KAOS is a generic methodology that is based on the capturing, structuring and precise formulation of the system goals [2]. A goal is a 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. In general, a goal may be regarded as a named formal property of the system that must be respected in the design and implementation phases.
- 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.
- 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 to break the high level goals of the system.

For brevity, we shall only concentrate in what follows on the use of the goal model to represent security properties and the use of the anti-goal model to express vulnerabilities undermining those 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 $G_1 \dots G_n$, then completeness and minimality are defined as follows:

- *Completeness*: $G_1 \wedge \ldots \wedge G_n \Rightarrow\ G$
- *Minimality*: $G \Rightarrow\ G_1 \wedge \ldots \wedge G_n$

Completeness states that when all of $G_1 \dots G_n$ 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. It is worth mentioning that our mathematical definition of minimality differs from the classical KAOS definition of [1].

In the anti-goal model [3], 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 obtain domain-specific vulnerabilities. Minimality plays an importan role here, since breaking a goal means that at least one of the subgoal has been broken.

3. 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 [5]. 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 right issued by the VO to access the data and
  - is 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.

3.1. 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.

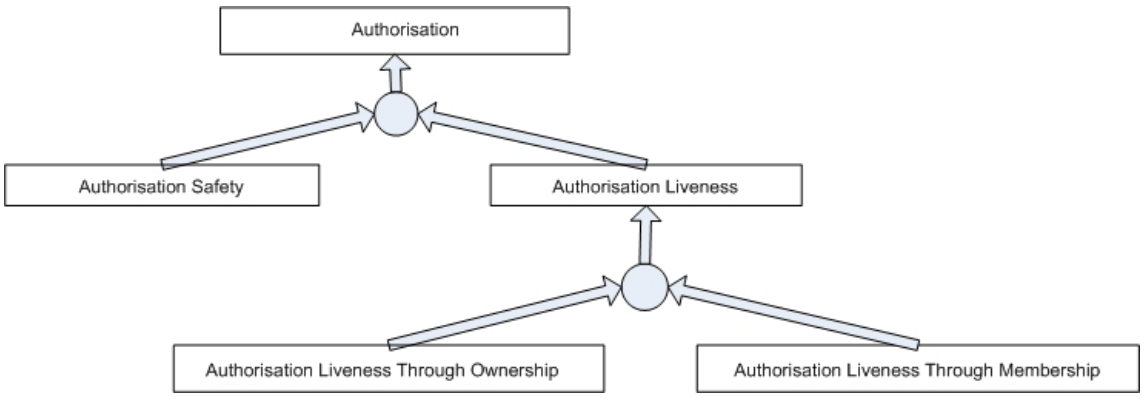

FIGURE 1: The Goal Model for Authorisation.

- *Data*: the set of all data including that which may be created and used by XtreemOS applications.
- *Principal*: the set of principals in a VO. Principals may be users, administrators or services present on the Grid.
- *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*: This is the set of credentials. A credential may be a password, a certificate or any other mechanism.
- *Attribute*: This is the set of attributes of a credential. This set includes data ownership, VO membership, task assignment and data read and write attributes.
- *Right*: is the set of access rights, *read* and *write*. Note that $Right \subset Attribute$.
- *owner* : $Principal \times Data \rightarrow \mathbb{B}$ is a predicate on principals and datasets denoting that a principal owns a dataset.
- *member* : $Principal \times VO \rightarrow \mathbb{B}$ is a predicate on principals and VOs 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 on tasks and datasets 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 definitions, 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 $\forall p \in \textit{Principal}, d \in \textit{Data}, r \in \textit{Right}$:
$\textit{authorised}(p, d, r) \Rightarrow$
$(\textit{owner}(p, d) \vee (\exists vo \in \textit{VO}, t \in \textit{Task} : \textit{member}(p, vo) \wedge \textit{hasVORights}(p, d, vo, r) \wedge$
$\textit{assigned}(p, t) \wedge \textit{requires}(t, d)))$

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

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 $\forall p \in \textit{Principal}, d \in \textit{Data}, r \in \textit{Right}$:
$\textit{authorised}(p, d, r) \Leftarrow \textit{owner}(p, d)$

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

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

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

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

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

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

The last definition may be specialised using the read/write credentials, as follows:

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

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

Additionally, we needs to state the well-definedness of ownership credentials, as follows:

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

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.

3.2. An Anti-Goal Model for Authorisation

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. Informally, the anti-goal model for the breach of authorisation is shown in Figure 2.

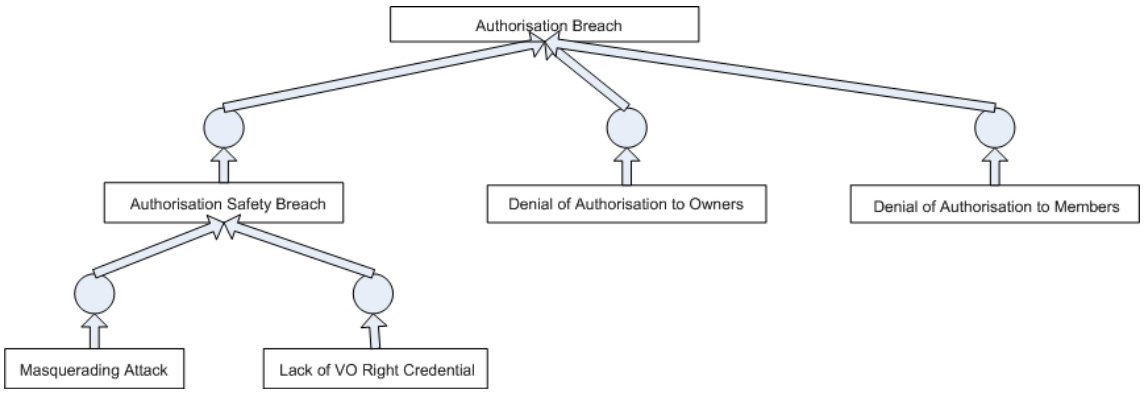

FIGURE 2: The Anti-Goal Model for Authorisation.

In this figure, 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 $\exists p \in \textit{Principal}, d \in \textit{Data}, r \in \textit{Right} : \ \Diamond \ (\textit{authorised}(p, d, r) \ \wedge \neg(\textit{owner}(p, d) \vee$
$(\exists vo \in \textit{VO}, t \in \textit{Task} : \ \textit{member}(p, vo) \ \wedge \ \textit{hasVORights}(p, d, vo, r) \ \wedge \ \textit{assigned}(p, t) \ \wedge \ \textit{requires}(t, d))))$

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

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

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

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

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

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. On the other hand, expanding with the definition of the lack of VO rights, $\neg \textit{hasVORights}(p, d, vo, r)$, yields the following anti-subgoal:

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

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, one which is a breach of confidentiality and the other a breach of integrity:

AntiGoal [Lack of VO Read Credential (Confidentiality Breach)]
FormalDef $\exists p \in \textit{Principal}, d \in \textit{Data}, r \in \textit{Right}, \forall cr \in \textit{Credential}, vo \in \textit{VO}, t \in \textit{Task}$:
$\Diamond$ (*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 [Lack of VO Write Credential (Integrity Breach)]
FormalDef $\exists p \in \textit{Principal}, d \in \textit{Data}, r \in \textit{Right}, \forall cr \in \textit{Credential}, vo \in \textit{VO}, t \in \textit{Task}$:
$\Diamond$ (*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*$(t, d)))$

3.3. 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 on in the system development such as masquerading attacks and the lack of VO rights.

4. CONCLUSION AND FUTURE WORK

To conclude, we have defined formal goal and anti-goal models for the autherisation security property in a grid-based operating system; XtreemOS. These models provide an insight into the security requirements of system and are essential in guiding the development of the XtreemOS system in its design, implementation and maintenance phases. Our approach has consisted in applying the KAOS goal-oriented RE methodology, which formalises goal specifications using linear temporal logic and includes the notion of goal refinement and goal operationalisation as well as techniques for reasoning about security goals.

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 well as remove ambiguity from their definitions.

As future work, we plan to extend our model to for other security requirements in XtreemOS, such as Single-Sign On (SSO) and communication security. We are also investigating the usefulness of our model in other Grid systems in order to determine if there are certain patterns of security requirements and their vulnerabilities that could be extracted from this work. Some open issues remains on the application of KAOS for security. Although there is notion of completeness for handling obstacles [4], hence anti-goals, it is desirable to have a notion of completeness covering all potential threats to the system. Finally, we are also working toward developing a technique for

formally deriving test-cases from the requirements model, which will be applied to the security critical components of the XtreemOS architecture.

REFERENCES

[1] R. Darimont and A. van Lamsweerde. Formal Refinement Patterns for Goal-Driven Requirements Elaboration. In *FSE-4 - 4th ACM Symposium on the Foundations of Software Engineering*. ACM, 1996.

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

[3] 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.

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

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