Analysing Access Control Specifications

Probst, Christian W.; Hansen, René Rydhof

Published in:
Proceedings of the Fourth International IEEE Workshop on Systematic Approaches to Digital Forensic Engineering

Link to article, DOI:
10.1109/SADFE.2009.13

Publication date:
2009

Document Version
Publisher's PDF, also known as Version of record

2009 Fourth International IEEE Workshop on Systematic Approaches to Digital Forensic Engineering

Analysing Access Control Specifications

Christian W. Probst
Technical University of Denmark
Email: probst@imm.dtu.dk

René Rydhof Hansen
Aalborg University
Email: rrh@cs.aau.dk

Abstract

When prosecuting crimes, the main question to answer is often who had a motive and the possibility to commit the crime. When investigating cyber crimes, the question of possibility is often hard to answer, as in a networked system almost any location can be accessed from almost anywhere. The most common tool to answer this question, analysis of log files, faces the problem that the amount of logged data may be overwhelming. This problems gets even worse in the case of insider attacks, where the attacker's actions usually will be logged as permissible, standard actions—if they are logged at all. Recent events have revealed intimate knowledge of surveillance and control systems on the side of the attacker, making it often impossible to deduce the identity of an inside attacker from logged data. In this work we present an approach that analyses the access control configuration to identify the set of credentials needed to reach a certain location in a system. This knowledge allows to identify a set of (inside) actors who have the possibility to commit an insider attack at that location. This has immediate applications in analysing log files, but also non-technical applications such as identifying possible suspects, or, beyond cyber crimes, picking the "best" actor for a certain task. We also sketch an online analysis that identifies where an actor can be located based on observed actions.

1. Introduction

At the very core of both businesses as well as society we nowadays find information systems that pervade our daily life. They are essential at all levels of actions, from meaningless "surfing" to decisions that potentially can influence the well-being of companies, economies, or even whole populations.

Not only do these information systems support everyday life and day-to-day operations, in many cases they enable them in the first place, and often indeed are among the most valuable assets of individuals or organisations. Offering seamless access to computing resources and data from virtually any location around the globe is one of the most outstanding features of these systems. This flexibility makes them valuable in the first place, but at the same time is also the reason for their major vulnerability—via the network, an entity's data is accessible from almost everywhere, often without the need of physical presence in the data's perimeter.

This risk of data being accessible without proper legitimation, has led to a wide range of access control mechanisms, which are supposed to restrict access to data. The standard approach to securing data is to tighten access control measures. When these measures do not serve their purpose, that is in case of a cyber crime, investigators often have to fall back on log file analysis to find out, what has happened, and who may have been the attacker.

When prosecuting crimes, the main question to answer is often who had a motive and the possibility to commit the crime. When investigating cyber crimes, the question of possibility is often hard

978-0-7695-3792-4/09 $25.00 © 2009 IEEE
DOI 10.1109/SADFE.2009.13

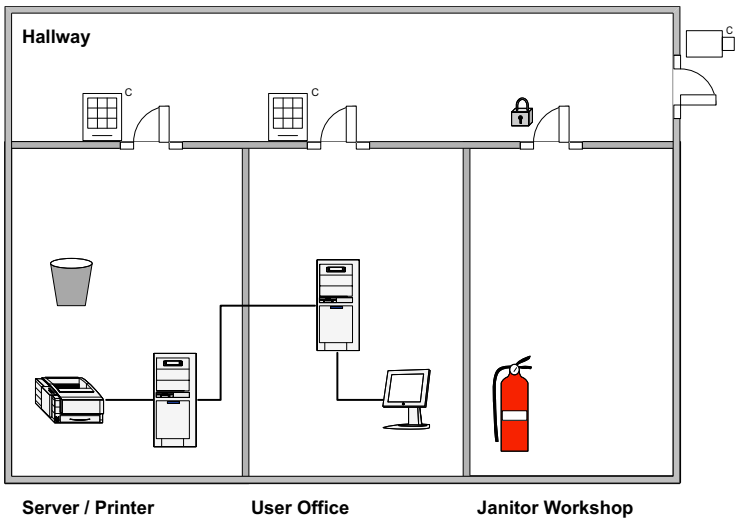

Figure 1. The example system used to illustrate our approach. Besides the "real" building, the computer network allows to access data virtually. The icons on the doors specify the kind of access control applied, *e.g.*, face recognition at the entrance, and a regular key to the janitor's workshop.

to answer, as in a networked system almost any location can be accessed from almost anywhere. The aforementioned, most common tool to answer this question, analysis of log files, faces the problem that the amount of logged data may be overwhelming. If the attacker is an insider instead of an outside attacker, analysing log files may turn out to be even harder, since the insider's actions usually will be logged as permissible, standard actions—if they are logged at all. Recent events have revealed intimate knowledge of surveillance and control systems on the side of the attacker [1,8,15], making it often impossible to deduce the identity of an attacker from logged data.

In previous work we have presented a system model that allows to model real-world organisations, integrating both the virtual and the real domain, that is, *e.g.*, buildings and networks [13]. Based on this model, we have presented analyses of possible behaviour of actors in the system, *e.g.*, which data they can access, or which locations they can reach [13, 14]. The latter analysis served the purpose of tightening the access control system to hinder actors from reaching certain areas of the modelled system, based on who the actor is, what he knows, and where he is (*identity*, *keys*, and *location*). The same work also presented an "after the fact" analysis, which analysed the system model and a log file together to identify who reached which locations in the system and accessed which data.

In this work we present an approach that combines these two analyses. It analyses the system model to collect the set of credentials that are needed to reach point A from point B, but without taking the log file into account. This goes beyond typical cyber-crime analyses, and aims at combing information about "the real world" with the virtual domain. We believe that this is important, also because it allows to help the analysis with events observed during investigations, or to experiment with assumed events to check their effect.

Since the knowledge computed by our analysis is present in the system model, the analysis can be seen as slicing of the system. It takes two locations and for each possible path between these computes the credentials needed to traverse it.

The analysis result allows to identify a set of (inside) actors who have the possibility to commit an insider attack at that location. This in turn can be used to guide the analysis of logged actions, allowing to focus the investigation.

Our analysis has also non-technical applications. In cyber crime investigations it can be used to identify the set of suspects who should be investigated or monitored. Especially identity-based access control rules allows to narrow down the set of suspects dramatically, depending how intrinsic

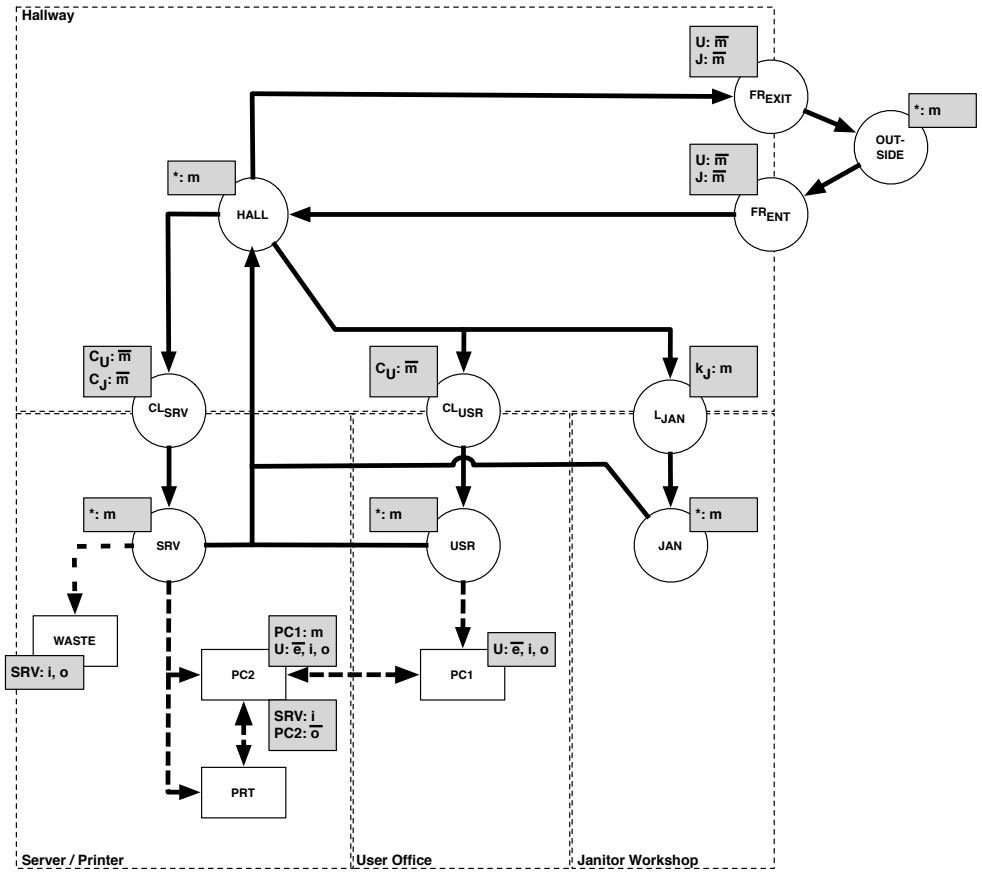

Figure 2. The abstract model of the example system from Figure 1, with policy annotations. The dashed boxes indicate the rooms from the original example. Each room is now replaced by all the locations it contains, $e.g.$, the user office has the cypher lock ($\mathtt{CL}_{\mathtt{USR}}$), a location for the office itself (USR), and the workstation (PC1). Circles identify these locations, and rectangles identify artifacts that the users cannot move to but only access from a neighbouring location. The different kinds of connections between locations indicate the physical domain (solid lines), the virtual domain (dashed), and access only (dotted). The policy annotations refer to two actors, janitor J and user U, who, $e.g.$, have different access rights to the user office and the server room. Each access control specification consists of the required key, identity, or location, and the permissible action. Actions are identified by their first letter, $e.g.$, m for move, i for input, etc. The overlined letters mark logged actions.

the identity-establishing factor is.

In the non-cyber crime area, our analysis has another, surprising by-product. Suppose that an organisation needs to send somebody from one point in the system to another, for example to pick something up. In this scenario our analysis determines the credentials required for this, and based on these all users possessing the necessary credentials can be identified. Out of this set of users one then can choose one randomly, or select the one who best fulfils a certain criterion, *e.g.*, having the least privileges, not being able to access a certain resource, etc.

This article is structured as follows. Before presenting the analysis, the rest of this sections introduces the example system used. This is followed by the presentation of the analysis in Section 3, and its application in Section 4. After discussing related work in Section 5, we conclude the paper in Section 6 and discuss future work.

1.1. Example

The analysis presented in this article works on any kind of system representing the structure of an organisation, consisting of both the physical locations as well as computer networks. As discussed in [13], any other similar structure could be added just as easily if needed for the sake of analysis, *e.g.*, the sewage system.

The floor plan of the example from [13, 14] is shown in Figure 1, the abstract model, which will be introduced in Section 2, is shown in Figure 2. The example consists of a simple building with a number of rooms connected by a hallway, and a computer network that connects two of the rooms.

In the example system we consider two actors, a general user U and a janitor J. For both of them access is restricted to some locations in the system. For example, each of the rooms has either a cipher lock or a regular lock, and access to the former is logged. Additionally, entrance to the whole system is restricted and logged, too.

2. System Model

In this section we present the system model used and the analysis we apply to it. The system model, based on a process calculus, provides the semantics of the analysis. The analyses, presented in the next section, are graph-based. The first one, given two locations in the system, traverses all possible paths between these locations and collects the set of credentials needed to traverse this path. The second analysis uses the results of another graph-based analysis, and allows to predict online at which locations an actor can be located.

The system model is based on previous work [13]. We only give a short overview here and refer to [13] for the details. The abstraction is based on a system consisting of components. We distinguish between location components, such as offices and computers, data components, such as keys and actual data, and mobile components, such as processes and actors. Data can be associated with (stored at) locations and actors, and it can be secured by, *e.g.*, encryption, and locations can be secured by access control mechanisms, *e.g.*, cipher locks. To support movements of dynamic components, locations can be connected by directed edges, which define freedoms of movements of actors.

A system consists of the *infrastructure*, which in turn consists of locations and connections between them. As shown in Figure 1, the model can contain different domains, where actors can only move in "their" domain, *e.g.*, processes can only access the computer network. *Actors* model everything that can move between locations and can perform *actions*. These actions consist of input, output, and moving, and in [13] are mapped to actions of acKlaim, a process calculus based on Klaim [10].

An important property of the system model is that it can be extended with new components. In [13] we present several of these extensions, such as logging, encryption and decryption of data, and access control. The latter consist of annotations at locations that specify when access to the location is allowed. These annotations are evaluated by a reference monitor in the operational semantics before the corresponding action is performed, similar to [6, 7, 14]. This is based on *capabilities* that actors can acquire, and *restrictions* that locations apply. Both restrictions and capabilities can be used to restrain the mobility of actors, by requiring, *e.g.*, a certain key to enter a location, or allowing access only for certain actors, or from certain locations.

In the example system in Figure 1 we could interpret the face of the actors as capability to enter the entrance of the building (based on face recognition), and the cipher keys as capabilities to enter the server room and the user office. The associated checkers would implement the test whether an actor with a given face is allowed to enter the building, or whether a cipher code matches the ones stored in the lock.

In Figure 2 the boxes specify which actions are allowed at the location they are associated with. As mentioned above we distinguish

- access based on *identity*, *e.g.*, when entering the system at $\texttt{FR}_{\texttt{ENT}}$,
- access based on *location*, *e.g.*, access to PC2, and

```
 1: extractCredentials(node n)
 2: /* return the credentials needed to move to node n */
 3: credentials = ∅
 4: acl = access control list at n
 5: for all restrictions (factor, actions) ∈ acl do
 6:   for all actions a in actions do
 7:     if a is move then
 8:       if factor is an actor or a key then
 9:         credentials = credentials ∪ {factor}
10:       end if
11:     end if
12:   end for
13: end for
```

Figure 3. The function *extractCredentials* returns a set of credentials needed to enter the node n. This can be either the identity of the actor or a key the actor needs to know.

- access based on *knowledge*, *e.g.*, access to $\mathtt{CL}_{\mathtt{SRV}}$.

Note that in this work we are mainly interested in the *move* action, which allows actors to change locations, and is used in access control to constrain movements in the underlying semantics.

The main property of our system model is that it is graph based, and therefor lends itself to the application of simple, graph-based analyses. While the underlying process calculus and its semantics can be used to analyse processes, which, *e.g.*, could describe observed behaviour of an actor. However, this is in general not of big interest when analysing log files or identifying potential threats, as we do not know the actions that will be performed, and therefor doe not know which process to analyse. This can be compared to analysing open systems, where our analysis results need to hold for every possible process we permit into the system [7, 11].

Another important property is that the model is modular, allowing to specify parts of the system individually, and plugging them together whenever a bigger system is required. This allows, for example, to combine models developed in different investigations.

3. Analysing Access Control Specifications

Having introduced our system model we are now ready to present the new analyses. In the following we assume a system of interest given by a specification like the one defined informally in the previous section, that is the system structure as a graph, and access control restrictions as annotations on the nodes.

The analyses in this section aim at two different goals as described above. The first analysis computes the set of credentials needed to traverse a certain path in the system model. As can be expected this is realised as a fairly simple, graph-based traversal of the system model. The second analysis is based on our previous work on log-equivalent locations. In contrast to the first analysis and the work in [13], this is an online analysis, that allows to dynamically trace locations where actors may be located.

3.1. Identifying required credentials

The goal of this analysis is to identify the credentials needed to move between two points in the system. This knowledge can then be used to guide the analysis of a log file when trying to find out who had possibility to commit an attack. When not considering log files, the same knowledge can of course also be used to narrow down or extend the set of suspects.

As mentioned above, our system model supports different mechanisms to constrain access; *identity*, *keys*, and *location*. At each location access can be granted because one or several of these mechanisms, and the analysis computes a sequence of locations and needed credentials to enter each location. Since this knowledge is also present in the graph, the analysis can be seen as slicing of the system graph. It takes two locations and for each possible acyclic path between these computes the credentials needed to traverse it. Cycles can be ignored since the analysis only depends on the

```
1: credentialsForPath(node from, node to)
2: /* return the credentials needed to move from node from to node to */
3: for all paths p = n_1, ..., n_k with k > 1, n_1 = from, n_k = to, and i ≠ j ⇒ n_i ≠ n_j do
4:   /* initialise IDs to contain all identities in the system */
5:   IDs_p = AllIDs
6:   result_p = ∅^k
7:   for all nodes n_i = n_2, ..., n_k do
8:     result_{n_i} = extractCredentials(n_i) \ (AllIDs \ IDs_p)
9:     IDs_p = IDs_p ∩ result_{n_i}
10:    if result_{n_i} = ∅ then
11:      result_{p,i} = ⊥
12:      continue
13:    else
14:      result_{p,i} = result_{n_i} \ ⋃_{1<j<=i} result_{p,j}
15:    end if
16:  end for
17: end for
18: return the computed tuples result_p and IDs_p
```

Figure 4. The function *credentialsForPath* returns for each path between two nodes a tuple of credentials needed to take that path. Whenever one or several identities are needed to enter a node, this is recorded in the global set *IDs* (see discussion in text). If the analysis at any point encounters a node that can not be entered, the according element in the tuple is set to $\perp$, and the next path is examined.

nodes reachable from another node based on knowledge the actor has, not how the actor reached that point. Therefore, reaching the same location again will not change the analysis result.

The pseudo code for this analysis is shown in Figure 3 and Figure 4. The function **extractCredentials** is used to identify for a node which credentials are needed to gain access. While this is fairly simple for our system model, this could involve arbitrarily complex functions. The same approach could also be used to identify the credentials needed to access some data.

This function is used by **credentialsForPath**, which visits all nodes on a path and computes a tuple with the same length as the path. For each node on the path the tuple contains the set credentials needed to enter this node. For example, a position with the empty set identifies a node that is not protected by access control, or that can be entered using only credentials already used before. Besides these sets the tuples can also contain a special marker $\perp$, identifying a node that can not be entered at all.

The latter situation can occur when on the path credentials are needed that are specified as being bound to a certain actor. For example, in a system requiring face recognition such as sketched in Figure 1, the face could be supposed to be an inseparable part of an actors identity. On a path $a^{\mathtt{J},\mathtt{U}:\mathbf{m}} \rightarrow b^{\mathtt{J}:\mathbf{m}} \rightarrow c^{\mathtt{U}:\mathbf{m}}$, an actor with identity J or U can enter the first node, but the next node can only be entered by J, and the last one only by U. If the actor's identity can not be separated from the actor, then there is no actor who could move from a to b. The analysis result for this small example would be the tuple $(\{J,U\},\{J\},\perp)$.

In the analysis we use the global parameter *IDs* to track identities; initially it is set to contain all identities in the analysed system, thus allowing easy tailoring of the analysis to different scenarios. Throughout the analysis this set is updated by intersecting it with the credentials needed to access the node analysed. As a result the set never gets bigger, as identities not able to enter the current node are removed. Once a node is met that can only be entered based on identities that are not part of the global set, the analysis stops.

The analysis just described, while being simple, allows to extract important knowledge from the access control configuration of a system. The path-based, global view on requirements of credentials make the overall control enforced more transparent than the local view at each location. One could imagine an inverse approach, where the system designed specifies which users should be able to get from where to where in the system (virtual or real), and an analysis similar to the one presented here is used to compute the necessary access control restrictions at the system's nodes. We leave this for future work. An application of the analyses presented in this and the next section will be

```
 1: equivalent()
 2: /* perform (log-)equivalent actions */
 3: changed = true
 4: while changed do
 5:    changed = false
 6:    for all actors n do
 7:       for all locations l that n might be located at do
 8:          for all locations l' reachable from l in one step do
 9:             simulate all actions that n can perform on l' (without causing a log entry in case of LTRA)
10:             for each action set changed if n at location l learns a new data item
11:          end for
12:       end for
13:    end for
14: end while
```

Figure 5. For each actor in the system we check for all locations he can be located at whether he can perform any actions. All these actions are assumed to have been performed. In the case of the log trace reachability analysis, only actions that would not cause a log entry are considered.

discussed in Section 4.

3.2. Online Prediction of Locations

While the analysis presented in the previous section is concerned with identifying the necessary credentials for passing a path based on an offline examination of a system's access control specification, we now turn to an online analysis with a slightly different focus. In [13] we have presented an analysis based on log-equivalent locations. That analysis takes a system specification and a log file, and identifies for each logged action all locations where an actor possibly can be. The latter identification is based on all locations that the actor can reach without causing a log entry.

A note on the notion of time used in our approach seems in order. We focus on modelling structural aspects of action traces with various logging policies. We therefore use a abstract notion of time that does *not* assume or rely on global synchronisation, but does guarantee a correct global ordering of events. This abstract notion of time is inspired by the notion of "Lamport time" [9]. This design choice allows us to consider problems relating to time and synchronisation separately, and independently of intrinsic structural problems. As noted by Lamport this global ordering is an idealistic abstraction that cannot achieved in systems of distributed clocks. Either one needs to order observed events manually, or one needs to take intervals of error into account. The former is trivial, though not always possible, and we are currently working on the latter.

In this section we extend on that analysis, by using its result to formulate an online analysis. While the former computes the sets of location where an actor might be, we now use that result to construct a state machine that keeps track of actors based on their observed actions. Observable actions are a central concept when designing secure systems, as they define what we can observe about our system. Similarly what we want to define influences which observables we need to implement.

3.2.1. Equivalent Locations and Actions

Before we present the online analysis we briefly recap the offline version that it is based on. As mentioned above, this is based on *equivalent locations and actions*, which from the viewpoint of an observer, in our case the analysis, can not be distinguished. Thus, if an actor can be in a location ℓ, then he might just as well be in *any* equivalent location, or might have performed any actions in between. In the log-trace reachability analysis presented in [13], log-equivalency is needed to find out what might have happened between two log entries.

The pseudo code in Figure 5 shows the realisation of log equivalency in the log-trace reachability analysis. It simply visits all locations where a user might be, and computes the effect of every unlogged action that the user is allowed to perform at that location. This computation is repeated until no further changes to the graph occur. The implementation of regular equivalency is quite similar, the only difference being that there is no restriction as to causing a log entry.

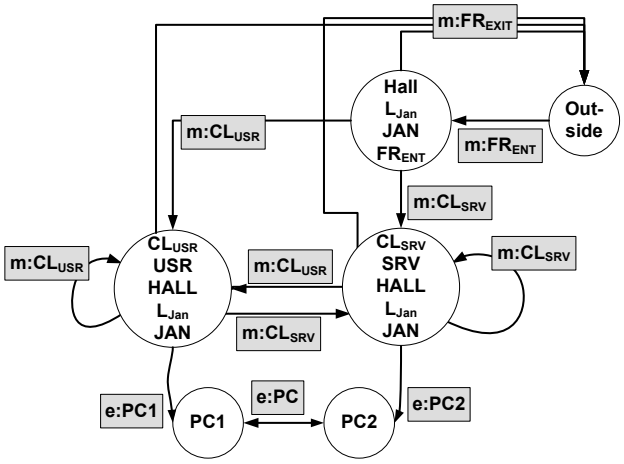

Figure 6. The model of observable transitions in the system from Figure 2. Each time a logged event is observed in the system, the automaton retraces the event, and based on the current positions of actors updates who is where.

3.2.2. Constructing a Finite Representation

The analysis in [13] computes for a log file which actors might have caused the entries, and where they can be located in the system. We now use this result to construct a finite automaton, that has sets of locations as states, and pairs of actions and locations on the edges.

At runtime or "system observation time", to be more precise, each actor is initially placed at a starting location. Whenever an action is observed, the finite automaton is used to decide which actor can be where and can have caused the action. This allows to identify threats in the system, and to guide the expectation of other logging and surveillance systems.

For the example system from Figure 2 the automaton is shown in Figure 6.

3.2.3. The Online Analysis

Having constructed the automaton, the online analysis will be able to predict positions of actors much better than the offline analysis described in [13]. The shortcoming of that analysis was that due to its offline nature it needed to make many assumptions about where actors might be. The online analysis, which we only sketch here, benefits from detailed knowledge about the locations of users.

For example, if a key is used somewhere in a system to get from A to B and then to C, then the static analysis has to assume that any actor who possibly was at A might have caused this sequence of events. On the other hand, if before the movement from B to C is observed, an action of user U is observed at B, then the online analysis can use this observation and identify U as the only possible user at C.

In the automaton as the one shown in Figure 6 we move all actors from node n to n' if an event e is observed at location l (as described by the edge labels). Whenever the analysis is able to identify precisely who caused an event, because the generated log event is based on an actor's identity, all user's that have not caused the event are removed from that location. This is similar to the narrowing performed in the offline analysis described in [13].

4. Application

The potential applications of our techniques are manifold, and go beyond digital forensics. In this area our analyses are similar to how investigators in "real-world" cases work. However, our techniques can go beyond these approaches. First, they can be used to get a starting point for

location	credentials
FRent	J, U
HALL	J,U
$\texttt{CL}_{\texttt{srv}}$	cU, cJ
$\texttt{CL}_{\texttt{usr}}$	cU
SRV	J,U
USR	J,U
PC1	U
PC2	U

Table 1. Credentials extracted for the nodes on the two example paths.

human investigations, by pre-computing reachable locations, accessible data, and other observations of interest. Furthermore, since our model is modular, investigators can easily combine them when either a connection between cases is identified, or at least might exist.

As a result of combining models, or in the case of modelling big and complex systems, the generated model can become equally big and complex, resulting in long analysis times. A main property of our approach is that, due to the underlying formal approach of static analysis [12], it is possible to collapse parts of the system. The solution for the resulting system will be less precise than for the whole system, but can be computed considerably faster. After the result for the partially collapsed system has been reviewed, one then can identify interesting parts, de-collapse them, and re-analyze to obtain a more detailed result.

In this section we first briefly present the application of the analysis described in Section 3.1 above to the example system. After that, we discuss the analysis of a more complex scenario in Section 4.2. We currently lack a meaningful complex model to illustrate the computational behaviour of our approach, but experiments with randomly generated graphs and policy annotations are very promising.

4.1. Required Credentials

We assume we want to obtain some data from PC1 in the example system, and the actors are located outside the system. The two possible paths are $p_1 = \texttt{Out} \rightarrow \texttt{FR}_{\texttt{ent}} \rightarrow \texttt{HALL} \rightarrow \texttt{CL}_{\texttt{srv}} \rightarrow \texttt{SRV} \rightarrow \texttt{PC2} \rightarrow \texttt{PC1}$ and $p_2 = \texttt{HALL} \rightarrow \texttt{FR}_{\texttt{ent}} \rightarrow \texttt{HALL} \rightarrow \texttt{CL}_{\texttt{usr}} \rightarrow \texttt{PC1}$. Since we are only dealing with two users, the set of all identities in the system is U, J.

Table 1 shows the credentials returned by the function **extractCredentials** for the different nodes on the paths. In what follows we step through the analysis of these paths. Since the first two locations are the same we obtain the same result for both paths, namely $result_{11} = result_{21} = \{J,U\} \setminus (\{J,U\} \setminus \{J,U\}) = \{J,U\}$ and $IDs_p = \{J,U\}$, and the same for the next node $result_{12} = result_{22} = \{J,U\} \setminus (\{J,U\} \setminus \{J,U\}) = \{J,U\}$ and $IDs_p = \{J,U\}$.

From here on the paths diverge. For the first path we obtain $result_{13} = \{cU, cJ\}\ (\{J,U\}\ \{J,U\}) = cU, cJ$ and $IDs_p = \{J,U\}$ and for SRV we get the same result $result_{14} = \{J,U\}$ and $IDs_p = \{J,U\}$. Finally, when accessing the location PC2, only the actor U is allowed to perform the action. This results in $result_{15} = \{U\} \setminus (\{J,U\} \setminus \{J,U\}) = \{U\}$ and $IDs_p = \{U\}$. The same results hold for the final step, when accessing PC1. For the resulting tuple this means that most positions are empty: $(\{J,U\}, \emptyset, \{cU, cJ\}, \emptyset, \emptyset, \emptyset)$ and $IDs_p = \{U\}$.

The second path diverges to the cypher lock for the user office, resulting in $result_{23} = \{cU\} \setminus (\{J,U\} \setminus \{J,U\}) = \{cU\}$ and $IDs_p = \{J,U\}$. For the next location we obtain $result_{24} = \{J,U\}$ and IDs_p remains unchanged. Finally, when accessing the location PC2, the same computation as for the first path results in $result_{25} = \{U\}$ and $IDs_p = \{U\}$. Consequently, the returned tuple for this path is the same as for the same path, as is the set of identities: $(\{J,U\}, \emptyset, \{cU\}, \emptyset, \emptyset)$ and $IDs_p = \{U\}$.

4.1.1. Identifying Suspects

While the current version of the example is rather unspectacular, it illustrates the analysis' purpose, namely to identify actors who can have caused a certain damage. Since only the user is

able to access PC1, it is clear who will be the culprit if a damage on that machine is found. On the other hand if something happens along the way, then the analysis also allows to identify all actors who possibly have been there.

4.1.2. Selecting Actors

Also the point of selecting actors can be illustrated with the example. While again it is fairly obvious that only U can access the location PC1 based on that location's access control specification, it also should be obvious that in more complex system such a decision will not always be so easy.

4.2. Analysing Complex Scenarios

Now we assume a more complex scenario in which we illustrate our method's modularity. Alice, Bob, and Charlie work in a company, and at some point confidential information of the company is leaked to a competitor. Once the leakage is detected, the server on which the document in question was stored is identified, and using the required credentials analysis shown in Section 4.1 the investigator finds out that all three actors had the necessary credentials to reach a location from which they could access the server. While the log entries show that Alice and Bob had entered the company area from which they could access the server, only Alice had permission to access the document. Consequently, she is suspected to have leaked its content.

Assume that in another investigation it is found that according to log files on a workstation Alice has logged into that machine; however, according to the results of the required credentials analysis he is not able to reach the room where the computer is located; the only employee able to do so is Charlie. This obviously indicates that Charlie is in possession of Alice's credentials.

Adding this knowledge as a fact to the analysis in the first case, we obtain the new result that Charlie or Alice might have leaked the document. Incidentally the investigator on the first case overhears that Alice has been stopped for speeding, and it turns out that this happened on the day the leaked document was accessed, and in a such a distance from the company, that Alice only could have been back at the company *after* the file got accessed. In our model this could be realised by adding a new location representing the stopping of Alice, and so many edges between the entrance to the company and the new location that they represent the time it takes to get back to the company. [1] This is similar to the process described above, where collapsed parts of the system (the node representing the "outside" world) are un-collapsed. Now an analysis of the log files as described in [13] will show that only Bob and Charlie where in the areal in question.

This last result has established that Alice cannot have been accessing the document. Together with the second result it can be shown that Charlie not only had the necessary credentials to masquerade as Alice, but also was in the area where the file was stored.

5. Related Work

This section gives a brief overview on related work and orthogonal approaches.

In the area of modelling, our approach is very similar to the Architecture Analysis & Design Language (AADL) [4], a modeling language that supports early and repeated analyses of a system's architecture with respect to performance-critical properties through an extendable notation, a tool framework, and precisely defined semantics. AADL is a quite powerful modelling language, with the goal of enabling modelling of complex electrical systems. As such it contains many mechanisms that go way beyond our abstract model. On the other hand, since our model is fairly simple and allows for straightforward modelling of communication systems, it should be embeddable into the AADL model. We are currently exploring this option, which would allow to use the quite natural support of process calculi for modelling, and then translate the model to an AADL model.

In the area of computer and network security, attack graphs [16] and privilege graphs [3, 5] are a widely used approach to model sets of actions that increase an adversaries capabilities. The

[1] It should be noted that in our current semantics all edges require constant time 1 to be passed. To model longer periods of time, several edges with dummy locations must be added.

graph can focus on whether a certain set of initial capabilities can eventually lead to some critical capability—this is very similar to actors obtaining new keys (or data items) in our analysis. However, there is a big difference; in attack graphs the edges are actions leading towards a goal (the attacker gains new capabilities) are away from it (the system administrator is able to disable some of the attackers capabilities). This requires to identify all potentially dangerous actions up-front; if an attacker is somehow able to obtain a key or knowledge about the system, this must be modelled in the attack graph. In contrast, we concentrate on the structure of the underlying system (edges), and analyse for data acquired by users. Combining these two techniques should allow for modelling distributed systems where users can obtain keys and knowledge in the non-virtual domains, thus acquiring new capabilities.

6. Conclusion and Future Work

Networked and interconnected information systems are at the very heart of our society. Not only do these information systems support everyday life and day-to-day operations, in many cases they enable them in the first place, and often indeed are among the most valuable assets of individuals or organisations. Offering seamless access to computing resources and data from virtually any location around the globe is one of the most outstanding features of these systems. This flexibility makes them valuable in the first place, but at the same time is also the reason for their major vulnerability—via the network, an entity's data is accessible from almost everywhere, often without the need of physical presence in the data's perimeter.

In this article we extend our previous work on analysing access control specifications of systems to guide and support forensics and investigation of cyber crimes. With the above mentioned systems getting more and more complex it also gets increasingly hard to see through the mesh of restrictions and permissions.

This is because the risk of data being accessible without proper legitimation, has led to a wide range of access control mechanisms, which are supposed to restrict access to data. The standard approach to securing data is to tighten access control measures. When these measures do not serve their purpose, that is in case of a cyber crime, investigators often have to fall back on log file analysis to find out, what has happened, and who may have been the attacker.

The analyses described in this article serve two purposes. First, they allow to gain a better understanding of capabilities of actors in a system. One obvious application is to identify which actors can have reached which locations in the system; using the system model sketched here and developed in [13], not only the physical but also virtual domains can be modelled. This of course is essential for analysing and preventing threats in networked systems, but also in the important area of pervasive networks.

As discussed above this analysis also has surprising by-products. Once one can analyse systems for reachability based on access control and actor's capabilities and keys, one can use the same approach to identify actors then are suited to perform a certain task. This selection can again be based on several criteria, depending on what the specific concerns are. These can range from "just get the job done", in which case we only want to pick somebody with sufficient rights to access all necessary locations; more restrictive would be requirements that aim at minimising the number of locations that can be accessed unnecessarily, or that disallow access to certain resources "along the way". The first analysis also allows to find actors that can only follow a specific way from all those that could be chosen—and there certainly are many more applications.

The second analysis also takes the access control specification into account, but it does so in two phases. First, the system is analysed offline to construct an automaton that retraces movements of actors based on observed events. These events can come from the logging and surveillance system, or they can come from various source as described, *e.g.*, in [2]. This automaton can then be used to observe logged actions, and thus allowing at any given time to predict which actors are located at which locations. This has obvious applications in surveillance, but also in preventing attacks, and in guiding forensics and investigations after attacks.

In this work we present an approach that combines these two analyses. It analyses the system

model to collect the set of credentials that are needed to reach point A from point B, but without taking the log file into account.

We are currently working on extensions of the analyses described here. It would be interesting to map the reachability analysis "backwards", that is to use it to *generate* access control specifications from required paths in the system. Another extension is the abstract domain in the first analysis presented—instead of restricting the tuples only based on identities one should also take the keys available to users into account, restricting these in similar ways as the identities.

On a system level we are investigating the mapping of our process-calculus based model into AADL, and to combine it with techniques like attack graphs and privilege graphs.

References

[1] Stein Bagger. Available from `http://en.wikipedia.org/wiki/Stein_Bagger`, last visited December 4, 2008.

[2] M. Beaumont-Gay, K. Eustice, V. Ramakrishna, and P. Reiher. Information protection via environmental data tethers. In *Proceedings of the New Securiy Paradigms Workshop (NSPW 2007)*, 2007.

[3] M. Dacier and Y. Deswarte. Privilege graph: an extension to the typed access matrix model. In *Proceedings of the European Symposium On Research In Computer Security*, 1994.

[4] P. H. Feiler, D. P. Gluch, and J. J. Hudak. The Architecture Analysis and Design Language (AADL): An Introduction. Technical report, CMU/SEI, 2006.

[5] Gorla and Pugliese. Resource access and mobility control with dynamic privileges acquisition. In *ICALP: Annual International Colloquium on Automata, Languages and Programming*, 2003.

[6] D. Gunnarsson. Static analysis of the insider problem. Master's thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby, 2007. Supervised by Christian W. Probst, IMM, DTU.

[7] R. R. Hansen, C. W. Probst, and F. Nielson. Sandboxing in myKlaim. In *The First International Conference on Availability, Reliability and Security, ARES'06*, Vienna, Austria, Apr. 2006. IEEE Computer Society.

[8] Jérôme Kerviel. Available from `http://en.wikipedia.org/wiki/Jerome_Kerviel`, last visited December 4, 2008.

[9] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[10] R. D. Nicola, G. Ferrari, and R. Pugliese. KLAIM: a Kernel Language for Agents Interaction and Mobility. *IEEE Transactions on Software Engineering*, 24(5):315–330, May 1998.

[11] R. D. Nicola, D. Gorla, R. R. Hansen, F. Nielson, H. R. Nielson, C. W. Probst, and R. Pugliese. From flow logic to static type systems for coordination languages. In D. Lea and G. Zavattaro, editors, *COORDINATION*, volume 5052 of *Lecture Notes in Computer Science*, pages 100–116. Springer, 2008.

[12] F. Nielson, H. R. Nielson, and C. L. Hankin. *Principles of Program Analysis*. Springer-Verlag, 1999.

[13] C. W. Probst and R. R. Hansen. An extensible analysable system model. *Information Security Technical Report*, 13(4):235–246, 2008.

[14] C. W. Probst, R. R. Hansen, and F. Nielson. Where can an insider attack? In T. Dimitrakos, F. Martinelli, P. Y. A. Ryan, and S. A. Schneider, editors, *Formal Aspects in Security and Trust*, volume 4691 of *Lecture Notes in Computer Science*, pages 127–142. Springer, 2006.

[15] N. D. Schwartz and K. Bennhold. A trader's secrets, a bank's missteps. *New York Times*, February 5, 2009.

[16] L. P. Swiler, C. Phillips, D. Ellis, and S. Chakerian. Computer-attack graph generation tool. *DARPA Information Survivability Conference and Exposition,*, 2:1307, 2001.