2014 IEEE Security and Privacy Workshops

Combining Generated Data Models with Formal Invalidation for Insider Threat Analysis

Florian Kammüller
Middlesex University London
f.kammueller@mdx.ac.uk

Christian W. Probst
Technical University of Denmark
cwpr@dtu.dk

***Abstract*—In this paper we revisit the advances made on invalidation policies to explore attack possibilities in organizational models. One aspect that has so far eloped systematic analysis of insider threat is the integration of data into attack scenarios and its exploitation for analyzing the models. We draw from recent insights into generation of insider data to complement a logic based mechanical approach. We show how insider analysis can be traced back to the early days of security verification and the Lowe-attack on NSPK. The invalidation of policies allows modelchecking organizational structures to detect insider attacks. Integration of higher order logic specification techniques allows the use of data refinement to explore attack possibilities beyond the initial system specification. We illustrate this combined invalidation technique on the classical example of the naughty lottery fairy. Data generation techniques support the automatic generation of insider attack data for research. The data generation is however always based on human generated insider attack scenarios that have to be designed based on domain knowledge of counter-intelligence experts. Introducing data refinement and invalidation techniques here allows the systematic exploration of such scenarios and exploit data centric views into insider threat analysis.**

***Keywords*-Insider threats, policies, formal methods**

I. Introduction

This work basically combines two approaches presented first at last year's Workshop on Research in Insider Threats [1]: the first paper by Glasser and Lindauer [3] provides generated, i.e., synthetic, data to use as input to scientific analysis, the second work by us [2] invalidates policies to analyze physical and workflow specifications of organizations to detect insider attacks. While the former approach uses modelchecking and interactive theorem proving techniques for automated support of the insider threat analysis, the latter approach implements ad hoc data generation mechanisms based on scientific results from human behaviour research, psychology, and counter-intelligence experience.

The current attempt now combines these two approaches by offering a way how the data centric approach of Glasser and Lindauer's generic data [3] can add more information to the invalidation approach of Kammüller and Probst [2] by adding generic data and thereby allowing refinement to unravel the schizophrenic world of the insider. As a homage to the pioneers of machine based analysis in communication security and as the source of inspiration to the synthesis presented in this paper, we begin by retracing the steps of the classical attack by Gavin Lowe on the Needham-Schroeder Public Key protocol (NSPK) [4] that coincidentally was also discovered by modelchecking [5]. We elaborate that the insider's attack in the NSPK attack becomes in fact discoverable by adding a refined view on the principals' data (Section II). Having thus established a key idea how refinement of principals' data could be used for insider threat analysis, we retrace the steps taken by Kammüller and Probst and show how model checking in combination with simple data refinement can be applied on a real insider attack case study – the Lottery Fairy that fakes a winning lottery ticket and dates it back after the draw (Section III). Data refinement acts as a lever to boost the rigorous qualities of state exploration with modelchecking to insider threat analysis. A progress that can now be applied as a hook-in to exploiting generation of data to investigate new ways how insider attacks can be discovered (Section IV). We close the paper with a critical discussion (Section VI).

II. The First Insider: Needham-Schroeder's Intruder

The attack on the Needham-Schroeder Public Key protocol has often been used as an example to show the superiority of formal techniques over good engineering practice. The attack is a result of a change of security assumptions about the communication context. Needham and Schroeder designed this protocol as one of the first cryptographic protocols a few years after the invention of public key cryptography. This was the time when the first email systems were installed, and the Internet was still in its infancy. It was safe to assume that the principals that participated in network communication protocols were part of some group of honest people who did not act as attackers simultaneously. Therefore, it is understandable that even in 1990 when the BAN logic [6] was conceived and logical analysis was for the first time applied to security protocols, the flaw in the Needham-Schroeder Public Key protocol (NSPK) still was not discovered. It was only five years later, in a world that already saw the advent of the Internet as a public and

DOI 10.1109/SPW.2014.45

anonymous communication space, that Lowe identified a seemingly obvious but crucial attack on NSPK. The attack does not need to break any cryptography and still allows the attacker to impersonate a member of the network. This attack is at the same time the first *insider* attack since it is based on the fact that a seemingly trustworthy participant of the group of principals acts simultaneously as attacker. To the best of our knowledge, this basic fact about the attack has been overlooked till now. Usually the attack is characterized as a classical man-in-the-middle attack – which is only half-true. We first briefly recapitulate the NSPK protocol and its attack to show that it is an insider attack. Moreover, we use the attack to illustrate that a refinement on the data can be used to make the attack discoverable and that this remedy also secures the protocol.

We use the short form of the Needham Schroeder Public Key Protocol (NSPK) originally published by [7]. The protocol is usually written as follows using public keys K_A, K_B known globally and their secret counterparts K_A^{-1}, K_B^{-1} establishing nonces N_A, N_B in the process of authentication.

$$\begin{aligned} A \rightarrow B &: \{N_A, A\}_{K_B} \\ B \rightarrow A &: \{N_A, N_B\}_{K_A} \\ A \rightarrow B &: \{N_B\}_{K_B} \end{aligned}$$

The originally published protocol gives rise to the well-known attack of [4].

The attack goes as follows. The insider I is a normal member of the network. Therefore, he has like any A and B also an address and his public key K_I – for which only he has the corresponding private key K_I^{-1} is known in the communication network. What is more is that I is a peer, i.e., principals do actually communicate with I. This fact is the main clue for the attack and also the reason why this is an insider attack. I must wait for A to request a communication with I, i.e. A sends a message according to step one of the protocol to I. Now, I sets up a parallel session, pretends to be A towards B initiating a second "Step One" this time from $I(A)$ to B, i.e. from I using B's sender address. We show the attack next using these notations but numbering the protocol steps according as $\alpha.i$ and $\beta.i$ for $i \in \{1..3\}$ for each of the parallel sessions.

$$\begin{aligned} &\alpha.1 & A \rightarrow I &: \{N_A, A\}_{K_I} \\ &\beta.1 & I(A) \rightarrow B &: \{N_A, A\}_{K_B} \\ &\beta.2 & B \rightarrow A &: \{N_A, N_B\}_{K_A} \\ &\alpha.2 & I \rightarrow A &: \{N_A, N_B\}_{K_A} \\ &\alpha.3 & A \rightarrow I &: \{N_B\}_{K_I} \\ &\beta.3 & I(A) \rightarrow B &: \{N_B\}_{K_B} \end{aligned}$$

Looking at the steps, we can see that I switches between the roles of I and $I(A)$ at his leisure. The attack appears to be some kind of man-in-the-middle attack but it is really only successful as such towards B. While A quite rightly believes that he speaks with I – as he intended to – B believes he speaks to A while he really speaks with I.[1] The NSPK attack is easily fixed by introducing the sender B in step 2.

$$\begin{aligned} A \rightarrow B &: \{N_A, A\}_{K_B} \\ B \rightarrow A &: \{N_A, N_B, B\}_{K_A} \\ A \rightarrow B &: \{N_B\}_{K_B} \end{aligned}$$

This fix of the attack has been also discovered by Lowe who found the attack in the first place a year later using modelchecking with CSP/FDR [5].

As discussed, the attack is an insider attack since it only works if A addresses I as a legal member of the network. The switch between the two roles I and $I(A)$ could be considered as acting schizophrenically if it would not be intentional directed to the purpose of impersonation. The fact that this switch is possible is another necessary and sufficient condition for the success. Translated into practical terms, the notion of $I(A)$ could be implemented by IP packets with a fake sender IP address (which is perfectly possible in the Internet protocol IP and is also known as *spoofing*).

The point that we want to make here is that a refinement of the actor data can reveal the attack. If we add the address to the protocol by refining the date, the attack becomes obvious. This is exactly what the remedy presented above does and it is this observation that we take as starting point for the suggested exploration of insider attacks in this paper.

III. Catching Naughty Lottery Fairies

Lottery scams are an interesting domain for the study of insider attacks. The probably most notorious lottery fraud is the 1980 Pennsylvania lottery scandal commonly known as the triple six fix [8]. A group of employees including the announcer of the daily number Nick Perry rigged a three digit game known as the Daily Number by weighting all the balls except the four and the six. The effect was that the outcome was almost certainly to be made of three digits out of 4 and 6. On 4. April 1980, the scheme succeeded when 666 was drawn but the unusual betting patterns alerted the officials. The scheme was uncovered, all chief insiders were sent to prison and most of the illegal winning money was never paid out. Other classical lottery frauds are ones where the lottery shop workers cheat clients who bring in a winning ticket and claim the prize themselves. These are also insider attacks, e.g., [9].

Another insider attack is the case of the lottery agent who used the access to the lottery ticket machine to produce a ticket with the winning number combination after the draw and dated it back to a time before the draw [10]. We analyze the workflow based invalidation process [2] based on this last

[1]Even though the attack seems only half way successful, it can be exploited. Assume B is a bank and I sends a message "transfer 1000 \$ from my account to that of I" after the attack using A's credentials.

attack. This case study suffices to recapitulate the working of the workflow based invalidation.

The workflow based approach to invalidation [2] uses a formal description of the security policy for invalidation by either (a) direct logical negation or (b) applying data refinement of the workflow data to the policy before negating it.

For the lottery example, a simple description of a security policy is described by the following conditions.

- A lottery ticket l wins if and only if the chosen number sequence $c(l)$ matches the draw d, i.e., [2]
$$\text{winning\_ticket } l \equiv (c(l) = d)$$
- The ticket's time stamp t_l must be strictly smaller than the time of draw t_d, i.e.,
$$\text{valid\_ticket } l \equiv (t_l < t_d)$$

Finally, the policy can be expressed simply as the conjunction of these two predicates mandatory for all cashable lottery tickets.

$$\text{policy } x \quad \equiv \quad \text{winning\_ticket } x \wedge \text{valid\_ticket } x$$

As the global policy we assume a concrete set of "winners" and then quantify the policy over all lottery tickets in this set. The operator $\mathbb{P}(S)$ denotes the powerset of a set S.

$$\begin{array}{lcl} \text{winners} & : & \mathbb{P}(\text{ticket}) \\ \text{Policy} & \equiv & \forall x \in \text{winners. policy } x \end{array}$$

The policy formalization above in terms of logic allows to invalidate them by direct logical negation. Since we want to analyze ways how a fake ticket can pass the policy we simply assume such a fake lottery ticket.

$$\textit{fake} \quad : \quad \text{ticket}$$

Then the following two simple properties express two ways of invalidating the policy.

$$\begin{array}{llcl} 1. & \text{policy } \textit{fake} & \Rightarrow & \text{winning\_ticket } \textit{fake} \\ 2. & \text{policy } \textit{fake} & \Rightarrow & \text{valid\_ticket } \textit{fake} \end{array}$$

The first of these trivial invalidation properties can be interpreted as faking a ticket that has a number matching the draw; the second can be in fact considered as a very abstract expression of the actual attack we want to illustrate. The fake ticket is "valid", i.e., has a timestamp that is before the time of drawing: $t_l < t_d$. As further elaborated in [2], the interactive theorem prover Isabelle/HOL [11] can be used in formalizing the above policy definition and proving lemmas that ascertain the above reasoning on the policy violation of a fake ticket (see Appendix VII-1). Despite the formal provability and a possible interpretation to the actual insider attack we investigate, we would like to explore the attack a little deeper.

[2]For simplicity of the exposition, we omit statements $c \sqsubseteq d$ for a partial order relation to express partial lottery wins.

Therefore, we consider the simple data refinement on the workflow data (the lottery ticket) to be applied to the policy: instead of simply considering an added on timestamp we additionally introduce a sequence number on lottery tickets. The coupling invariant for the refinement introduces a consistency condition that enforces that tickets in the draw must have sequence numbers below *last-ticket* which represents the last issued one.

$$\begin{array}{lcl} \text{winners} & : & \text{sequence(ticket)} \\ \text{backdated} c & \equiv & \text{index } c < \text{index } \textit{last-ticket} \end{array}$$

Now, the invalidation of the refined policy `Policy_seq` (see Appendix) for sequences of tickets shows the attack more clearly in its antecedent:

```
∃ x mem l. backdated l ⟹ ¬ (Policy_seq l)
```

IV. Using Data Refinement to Find Dropbox Leakage

Glasser and Lindauer consider the generation of insider threat data using a synthetic data generation framework. They can avoid any confidentiality and privacy concerns related to real data. The realism of the data is an arguable point but to some extent it is achievable and the data can be economically generated, tuned to desired characteristics, and is fully intact. We use the data on the one hand side to validate our invalidation approach and simultaneously investigate the mutual benefits to the exploration of insider threat scenarios that may result. As Glasser and Lindauer exhibit [3, Section IV] a remaining challenge is the Insider Threat construction.

Although the input to the data generation process is largely autonomous and produces intelligent near realistic data an important ingredient are the basic insider scenarios that are manually inserted into the process. These insider scenarios are constructed using counter-intelligence expert knowledge. An example from [3] is the following: "A member of a group decimated by layoffs suffers a drop in job satisfaction. Angry at the company, the employee uploads documents to Dropbox, planning to use them for personal gain." The data generation process [3] derives so-called "observables" from this scenario. For the example, the observables are given in the following list [3] .

- Data streams end for laid-off co-workers, and they disappear from the LDAP directory.
- As evidenced by logon and logoff times, subject becomes less punctual because of a drop in job satisfaction.
- HTTP logs show document uploads by subject to Dropbox.

We illustrate first how invalidating policies based on system models serves to unravel insider attacks based on the example. For the formalization of the system models, we use ExASyM [12] a formalism dedicated to insider threat analysis. The analysis is performed with the MCMAS modelchecker [13] because this tool is specialized for multi agent systems. A systematic translation transforms ExASyM models to the input language of MCMAS. For the invalidation, we specify the goal as the negated policy and then apply modelchecking to find a path of ExASyM actions leading to a state that violates the policy.

In our model we have an actor `User` that has access to proprietary company data and behaves within the boundaries of the company policy and we have an actor `Insider` that corresponds to a malicious insider but with reduced access rights. We construct this model now intentionally so simple that it will not show any vulnerability to an attack of the insider. But the refined model produces the attack by invalidation. The `User` handles confidential and unclassified files and can put both of them on a server. The `Insider` can also handle files on the server but only unclassified one. In addition, the `Insider` can take files outside. In this simple model the invalidation attempt to show that confidential files get outside is never reached. The following formula checks `true` for the simple model in MCMAS.

```
AG!(Intruder.has_secretfile);
```

This formula reads "for all paths in all states the formula `Intruder.has_secretfile` is not true" (exclamation mark denotes negation).

However, the system model based approach only tests for attacks that have already been anticipated in the model, i.e., is confirmatory according to Glasser and Lindauer's terminology [3]. They observe that most generation methods for insider attacks suffer from that weakness and admit that their otherwise very successful approach does so as well. They use prefabricated human-master-minded scenarios as kernels for their generation process.

To examine this issue, we combine the strength of modelchecking with the expressivity of the generated data. In fact, introducing additionally refinement as in the lottery example above enables insertion of analytic data into the model to reveal the insider. The refinement of the model uses the observation from Section II that insiders act like having double personalities.

To illustrate the advantages of the combination of the generated data with invalidation, we use the Dropbox scenario showing how it can be produced as a refinement from the above system model based attempt. We additionally assume that there is a Dropbox to which the Insider and the User have access. The attack is now possible because the User actually copies a confidential file into the Dropbox. Now, the Insider can take the confidential file outside, i.e., access it at home for future exploitation. The sudden change of behaviour of the User is a consequence of his divided personality. To model this formally, and enable the attack, we introduce a special variable `UasI` that enables the coupling of the behaviours between User and Intruder. Given this refinement of the actor behaviour we can now verify the formula

```
EF(UasI -> Intruder.has_secretfile);
```

stating that there are states in which confidential data is put outside.

Summarizing, we achieve thus certainly that our analysis can be refined to the point where the model corresponds to the generic data as produced in the approach by Glasser and Lindauer [3]. Critically reflecting, we see that the refinement has been strongly directed towards the existing scenario. Nevertheless, the insertion of the double personality has been a schematic procedure learned by example from the NSPK attack. In other words, the exploration of model refinements is a good way of acquiring a realm of typical data or action refinements that can be applied on generated data thereby moving a bit closer to actually finding attacks.

V. Related Work

Santen [14] examines preservation of security under refinement. In general, security is not preserved by refinement, a property often referred to as the security refinement paradox. Invalidation actually exploits this restriction of refinement. Nevertheless, considering certain types of probabilistic information flow security allows security preserving refinement relations.

Security protocol analysis with modelchecking tools is a well explored area of security research. AVISPA [15] and ProVerif [16] are just two examples of recent successful tools. The modelchecking approach implements a complete state exploration of a property on a model. Thus a successful check of a (positive) security property over a protocol corresponds to a mathematical proof of that property with respect to the used model. Usually, it is the goal of modelchecking to positively prove a (security) property but the approach is equally useful for detection of flaws. This fault detection is often used, for example, in the early application by Lowe [5] we discussed in Section II and it is also this second use we mainly consider useful for invalidation. The CTL temporal logic quantifier `EF` p ("there exists a path such that eventually p") is particularly suited as it produces a path leading to a state in which property p is violated. This path then allows the construction of an attack path.

Edmonds [17] addresses simulation models that are similar to the generated data model we use for our analysis. He observes that it is in general very difficult – if not hopeless – to develop reliable models for social phenomena that could be useful for simulation. However, he argues that simulation modeling can be used to "boot-strap" useful knowledge about social phenomena. His line of argument is very similar to our approach: " If each bit of simulation work can result in the rejection of some of the possible processes in observed social phenomena [...] then this can be used as part of a process of gradually refining our knowledge about such processes in the form of simulation models."

VI. Conclusions

In this short paper, we have summarized previous work on insider threat analysis and generation of insider data, recapitulated its workings and shown that analysis results can help to refine the insider attack scenarios that are at the core of the generation method. We have contributed thereby to a broader understanding of insider threats in general. We have further illustrated that the NSPK attack is an insider attack and how to use this insight to facilitate the construction of the analysis model for the system model based invalidation approach.

The invalidation technique has proved again a useful method for analysis of system models, workflow models, and combinations thereof. The additional use of refinement techniques is beneficial. Data refinement has long been identified as a central method for developing specifications into implementations [18]. In the lottery example we have used a standard data refinement macro: "sets to sequences" to discover the backdating. In the combination with generated data, we use a mix of data and action refinement to exhibit the insider attack in the second model. The construction of this second refinement did use the general idea of the double personality of the insider. The way we wove this idea into the model to achieve the refined model, is rather ad hoc. It is an open question, whether we can extract a more schematic way for constructing a "double personality" refinement for random applications. It is worth investigating this particular refinement – maybe more than others – as it is characteristic for insider attacks.

VII. Acknowledgments

Part of the research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement #318003 (TREsPASS). This publication reflects only the author's views and the Union is not liable for any use that may be made of the information contained herein.

References

[1] *Proceedings of the second IEEE Workshop on Research in Insider Threats, WRIT'13.* IEEE, 2013.

[2] F. Kammüller and C. W. Probst, "Invalidating policies using structural information," in *WRIT'13.* IEEE, 2013.

[3] J. Glasser and B. Lindauer, "Bridging the gap: A pragmatic approach to generating insider threat data," in *WRIT'13.* IEEE, 2013.

[4] G. Lowe, "An attack on the needham-schroeder public-key authentication protocol," *Information Processing Letters*, vol. 56, no. 3, pp. 131–133, 1995. [Online]. Available: http://web.comlab.ox.ac.uk/oucl/work/gavin.lowe/Security/Papers/NSPKP.ps

[5] ——, "Breaking and fixing the needham-schroeder public-key protocol using fdr," in *Tools and Algorithms for the Construction and Analysis of Systems.* Springer-Verlag, 1996, pp. 147–166.

[6] M. Burrows, M. Abadi, and R. Needham, "A logic of authentication," *ACM Transactions on Computer Systems*, vol. 8, pp. 18–36, 1990.

[7] R. M. Needham and M. D. Schroeder, "Using encryption for authentication in large networks of computers," *Communications of the ACM*, no. 21, 1978.

[8] "1980 pennsylvania lottery scandal," Available from http://en.wikipedia.org/wiki/1980_Pennsylvania_Lottery_scandal, last visited February 5, 2014.

[9] "Shop workers con customer out of 1 million winning lottery ticket," Available from http://www.mirror.co.uk/news/world-news/1million-lottery-scam-father-son-2848818 last visited February 5, 2014.

[10] P. G. Neumann, *Combating Insider Threats.* Springer, 2010, ch. in [19].

[11] T. Nipkow, L. C. Paulson, and M. Wenzel, *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, ser. LNCS. Springer-Verlag, 2002, vol. 2283.

[12] C. W. Probst and R. R. Hansen, "An extensible analysable system model," *Information Security Technical Report*, vol. 13, no. 4, pp. 235–246, Nov. 2008.

[13] A. Lomuscio, H. Qu, and F. Raimondi, "Mcmas: A model checker for the verification of multi-agent systems," in *CAV*, ser. Lecture Notes in Computer Science, A. Bouajjani and O. Maler, Eds., vol. 5643. Springer, 2009, pp. 682–688.

[14] T. Santen, "Preservation of probabilistic information flow under refinement," *Inf. Comput.*, vol. 206, no. 2-4, pp. 213–249, 2008.

[15] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P.-C. Heám, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron, "The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications," in *Proceedings of the 17th International Conference on Computer Aided Verification (CAV'05)*, ser. LNCS, K. Etessami and S. K. Rajamani, Eds. Springer, 2005, vol. 3576, available at http://www.avispa-project.org/publications.html.

[16] B. Blanchet and B. Smyth, *ProVerif 1.85: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial*, 2011.

[17] B. Edmonds, "Bootstrapping knowledge about social phenomena using simulation models," *Journal of Artificial Societies and Social Simulation*, vol. 13, no. 1, p. 8, 2010. [Online]. Available: http://jasss.soc.surrey.ac.uk/13/1/8.html

[18] J. He, C. A. R. Hoare, and J. W. Sanders, "Data refinement refined," in *ESOP*, ser. Lecture Notes in Computer Science, B. Robinet and R. Wilhelm, Eds., vol. 213. Springer, 1986, pp. 187–196.

[19] C. W. Probst, J. Hunker, D. Gollmann, and M. Bishop, Eds., *Insider Threats in Cybersecurity.* Springer, 2010.

Appendix

Appendix VII-1, gives the Isabelle/HOL formalization for the lottery insider attack and the used refinement. Appendix VII-2 and VII-3 provide the MCMAS specifications for the Dropbox example: the simple model and the refined one that exhibits the attack.

1) Lottery Fraud in Isabelle:

```
theory LotteryScam
imports Main
begin
datatype number  = Number nat
datatype time = Time nat
datatype ticket = Ticket time number
consts d, t_d :: nat
consts winners :: ticket set
consts fake :: ticket

primrec get_time :: time ⇒ nat ("t_")
where  t(Time t) = t
primrec get_number :: number ⇒ nat  ("#(_)")
where #(Number n) = n

definition valid_ticket :: ticket ⇒ bool where
valid_ticket x ≡ t(ticket) < t_d
definition winning_ticket :: ticket ⇒ bool where
winning_ticket x ≡ #(t) = d
definition policy :: ticket ⇒ bool where
policy x ≡ wining_ticket x ∧ valid_ticket x

lemma invalidate_one: policy fake ⟹ winning_ticket fake
by (simp add: policy_def)
lemma invalidate_two: policy fake ⟹ valid_ticket fake
by (simp add: policy_def)

definition Policy :: ticket set ⇒ bool where
Policy l ≡ ∀ x ∈ l. policy x

lemma Policy_empty: Policy {}
by (simp add: Policy_def)

(* Refinement for Invalidation *)
consts winners_seq:: ticket list
definition backdated :: ticket ⇒ bool where
backdated x ≡ t(ticket) < t_d
definition Policy_seq :: ticket list ⇒ bool where
Policy_seq l ≡ ∀ x. x mem l ⟶ (¬ backdated x ∧ policy x)

lemma no_backdated_in_seq_model: ∃ x mem l. backdated x
                                ⟹ ¬ (Policy_seq l)
by (simp add: Policy_seq_def)

definition policy_refinement:: check list ⇒ bool where
policy_refinement l ≡ Policy(set l) ⇒ Policy_seq l

lemma invalidation_by_refinement_gen:
"∀ s. finite s ∧ s ≠ {} ⟶ (∃ l. set (l) = s ∧ Policy s
⟶ ¬(policy_refinement l))"
apply (rule allI, rule impI)
apply (erule conjE)
apply (drule Finite_set_list)
apply (erule exE)
apply (rule_tac x = "hd (l) # l" in exI)
apply (rule impI)
apply (simp add: policy_refinement_def)
apply (rule_tac c = "hd l" in no_backdated_in_seq_model)
by (simp add: hd_lem)
end
```

2) Simple Generated Data in MCMAS:

```
Agent User
Vars:
  initialposition : { office };
  currentposition : { office, server };
  work_confidential: boolean;
end Vars
  Actions = { work, pause, save_secret, save_public };
Protocol:
  currentposition = office : {work, pause, save_secret};
  currentposition = server : { save_public};
end Protocol
Evolution:
  currentposition = office if (currentposition = server
    and (Action = pause or work_confidential = true));
  currentposition = server if (currentposition = office
    and (Action = work and work_confidential = false));
end Evolution
end Agent


Agent Insider
Vars:
  initialposition : { outside };
  currentposition : { server, outside };
  has_secretfile: boolean;
end Vars
  Actions = { access_server, copy, logout };
Protocol:
  currentposition = outside: access_server;
  currentposition = server : {copy};
  currentposition = server : { logout};
end Protocol
Evolution:
  currentposition = server if (currentposition = outside
                          and Action = access_server);
  currentposition = server if (currentposition = server
                          and Action = copy);
  currentposition = outside if (currentposition = server
                            and Action=logout);
  has_secretfile = true if (currentposition = server and
                            User.Action = save_public);
end Evolution
end Agent
```

3) Refined Generated Data in MCMAS:

```
Agent User
Vars:
  initialposition : { office };
  currentposition : { office, server };
  work_confidential: boolean;
  UasI: boolean;
end Vars
  Actions = {work, pause, save_secret,
             save_public, save_dropbox};
Protocol:
  currentposition = office : {work, pause, save_secret};
  currentposition = server : {save_public, save_dropbox};
end Protocol
Evolution:
  currentposition = office if (currentposition = server
      and (Action = pause or work_confidential = true));
  currentposition = server if (currentposition = office
                 and (Action = work and
                 (work_confidential = false or UasI)));
end Evolution
end Agent
```

```
Agent Insider
Vars:
  initialposition : { outside };
  currentposition : { server, outside };
  has_secretfile: boolean;
end Vars
  Actions = {access_server, copy, logout};
Protocol:
  currentposition = outside: {access_server};
  currentposition = server : {copy};
  currentposition = server : {logout};
end Protocol
Evolution:
  currentposition = server if (currentposition = outside
                            and Action = access_server);
  currentposition = server if (currentposition = server
                            and Action = copy);
  currentposition = outside if (currentposition = server
                              and Action=logout);
  has_secretfile = true if (currentposition = server and
                         (User.Action = save_public or
                          User.Action = save_dropbox);
end Evolution
end Agent
```