Analysis of Integrity Policies using Soft Constraints

Stefano Bistarelli*
Dipartimento di Scienze, Universitá di Pescara , Italy
bista@sci.unich.it
IIT, CNR, Pisa, Italy
stefano.bistarelli@iit.cnr.it

Simon N. Foley†
Department of Computer Science,
Cork, Ireland.
s.foley@cs.ucc.ie

Abstract

An integrity policy defines the situations when modification of information is authorized and is enforced by the security mechanisms of the system. However, in a complex application system it is possible that an integrity policy may have been incorrectly specified and, as a result, a user may be authorized to modify information that can lead to an unexpected system compromise. In this paper we outline a scalable and quantitative technique that uses constraint solving to model and analyze the effectiveness of application system integrity policies.

1 Introduction

The 2001 Computer Crime and Security Survey [8] reported that 196 of the respondents to the survey could quantify their losses due to unauthorised use of computer systems at a total of US$378 million in the previous year. While access-control mechanisms, firewalls and so forth may help counter such losses, we can never be confident about security unless we are provided with some assurance of their effectiveness. Such assurance may be achieved, in part, by analysing whether a formal description of the system upholds certain security properties.

Conventional security models such as [3, 7, 14, 16] are operational in nature in that they define *how* to achieve integrity but do not define *what* is meant by integrity. For example, the Clark-Wilson model [7] recommends that well-formed transactions, separation of duties and auditing be used to ensure integrity. However, the model does not attempt to define whether a particular security policy configuration actually achieves integrity: evaluating a system according to the Clark-Wilson model gives a confidence to the extent that good design principles have been applied. However, when we define a complex separation of duty policy, we cannot use the model to assure that a user of the system cannot somehow bypass the intent of the separation via some unexpected circuitous route.

In [9] it is argued that to provide such assurances it is necessary to model the behavior of both the system (with its protection mechanisms) and the *infrastructure* in which the system operates. Infrastructure is everything that serves the system requirements: software, hardware, users, and so forth. Even if a system is functionally correct, the infrastructure is likely to fail: software fails, users are dishonest, do not follow procedures, and so forth. The system and its security mechanisms must be designed to be resilient to these infrastructure failures. Only when a system is characterized in this way can it become possible to analyze whether a particular system configuration (including security policy) ensures integrity.

The approach in [9] provides a formal trace based semantics for integrity that requires detailed formal specifications to be provided for the system and its infrastructure. This requires considerable specification effort and the cost of such in-depth specification and subsequent analysis may be justified for small critical security mechanisms. However, we conjecture that such integrity analysis would not scale well to the configuration of a large and/or complex application system because it would be necessary to formally specify and reason about the potential behavior of *every* infrastructure component, user and so forth. Furthermore, [9] does not consider any approach to mechanizing the analysis and formal verification process.

In this paper we outline a more abstract and complementary approach to [9] that requires less semantic detail about the operation of the system and its infrastructure. Rather than attempting to model the complete behavior of the system and infrastructure (as in [9]), we model only those components that are considered relevant to the security policy and configuration. This is done by modeling the system

*Supported in part by MIUR project "Constraint Based Verification of Reactive Systems" (COVER), and by MIUR project "Network Aware Programming: Object, Languages, Implementation" (NAPOLI).
†Support received from Science Foundation Ireland under Grant 00/PI.1/C075

and infrastructure in terms of the *constraints* that they impose over security relevant components of the system. This results in a definition of integrity consistency that can be solved as a constraint satisfaction problem [12].

An advantage to expressing integrity analysis as a constraint satisfaction problem is that there exists a wide body of existing research results on solving this problem for large systems of constraints in a fully mechanized manner, for example, [13]. Constraints have been used in many practical analysis tools, such as Concurrent Engineering and Computer Aided Verification, for example, [6]. Thus, the results in this paper provide a direction for the development of practical tools for integrity analysis of complex application system security policies.

A further advantage to using a constraint based framework is that it becomes possible to carry out a quantitative analysis of integrity using *soft constraints* [4,5] A quantitative analysis provides a fine-grained measure of how secure a system is, rather the simple coarse-grained false/true provided by the conventional 'crisp' constraints.

2 Introduction to Constraint Solving

Constraint Solving is an emerging software technology for declarative description and effective solving of large problems. Many real life systems are analized and solved using constraint related technologies.

Constraints have been sucessfully used in the analysis of a wide variety of problems ranging from network management, for example [10], to complex scheduling such as [2]. They have also been used to analyze security protocols [1] and in the development of practical security administration tools [11]. In [11] constraints are used to help the System Administrator to easily describe network configurations and relations among servers, firewalls and services for the final users. Constraints are used to represent, in a declarative manner, the relations among network objects. This permits the use of local propagation techniques to reconfigure the network when hardware/software changes occur (particularly in a wireless environment). Such automatic reconfiguration would not be possible if the network policy was encoded using conventional shell scripts.

The constraint programming process consists of the generation of requirements (constraints) and solution of these requirements, by specialized constraint solvers. When the requirements of a problem are expressed as a collection of boolean predicates over variables, we obtain what is called the *crisp* (or classical) Constraint Satisfaction Problem (CSP). In this case the problem is solved by finding any assignment of the variables that satisfies all the constraints.

Sometimes, when a deeper analysis of a problem is required, *soft* constraints are used instead. Soft constraints associate a qualitative or quantitative value either to the entire constraint or to each assignment of its variables. More precisely, they are based on a semiring structure $S = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ and a set of variables V with domain D. In particular the semiring operation $\times$ is used to combine constraints together, and the $+$ operator for projection.

Technically, a *constraint* is a function which, given an assignment $\eta : V \to D$ of the variables, returns a value of the semiring. So $\mathcal{C} = \eta \to A$ is the set of all possible constraints that can be built starting from S, D and V (values in A are interpreted as level of preference or importance or cost). Using the the levels, we can order constraints: to say that c_1 is better then c_2 we will write $c_1 \sqsupseteq c_2$.

When using soft constraints it is necessary to specify, via suitable combination operators, how the level of preference of a global solution is obtained from the preferences in the constraints. The combined weight of a set of constraints is computed using the operator $\otimes : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ defined as $(c_1 \otimes c_2)\eta = c_1\eta \times_S c_2\eta$. Moreover, given a constraint $c \in \mathcal{C}$ and a variable $v \in V$, the *projection* of c over $V - \{v\}$, written $c \Downarrow_{(V-\{v\})}$ is the constraint c' s.t. $c'\eta = \sum_{d \in D} c\eta[v := d]$.

3 Integrity Analysis with Crisp Constraints

In [9] functional requirements are expressed as properties over traces of actions at an interface of a system. We take a more abstract approach by describing requirements in terms of constraints on variables that are invariant over the lifetime of the system. In this section we re-cast the example in [9] in terms of these constraint variables.

Consider a simple enterprise that receives shipments, and generates associated payments for a supplier. Requirements Analysis identifies the actions *shipnote* and *payment*, corresponding to the arrival of a shipment (note) and its associated payment, respectively. For the purposes of integrity, the analysis has identified a requirement that the system should not pay its supplier more than the stated value of goods shipped.

Let the constraint variables ship and pay represent the *total* value of goods shipped to date and the total value of payments made to date, respectively. Constraint $\mathsf{Probity}$ describes the requirement as an invariant over variables ship and pay.

$$\mathsf{Probity} \equiv \mathsf{pay} \leq \mathsf{ship}$$

Consider the implementation depicted in Figure 1 that uses separation of duties [7] to ensure that $\mathsf{Probity}$ is met. When a shipment arrives a clerk verifies the consignment at goods-inwards (entering *consign* into the system). When an invoice arrives, a different clerk enters details into the system, and if the invoice matches a consignment, a payment is generated. So long as the operations are separated then a single clerk entering a bogus consignment or invoice can be detected by the system.

Let variables inv and con represent the total value of invoices and consignments, respectively, received to date.

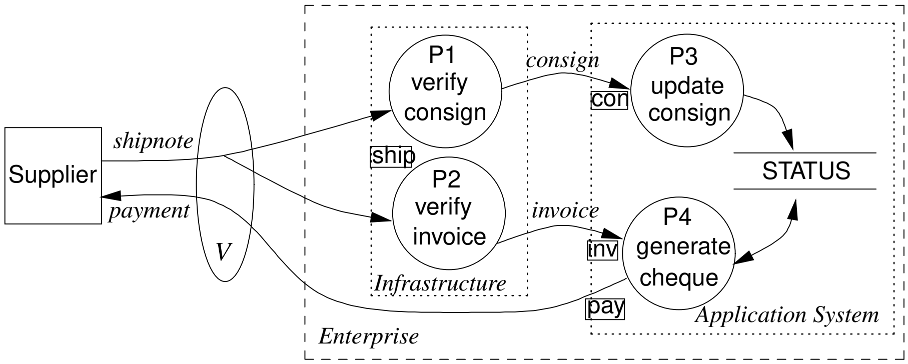

Figure 1. Supporting separation of duties

Specifications $\mathsf{Clerk1}$, $\mathsf{Clerk2}$ and Appl define the constraints on the system variables, reflecting invariants that are expected to be upheld by the clerks and the application.

$$\begin{aligned} \mathsf{Clerk1} &\equiv \mathsf{con} \leq \mathsf{ship} \\ \mathsf{Clerk2} &\equiv \mathsf{inv} \leq \mathsf{ship} \\ \mathsf{Appl} &\equiv \mathsf{pay} \leq min(\mathsf{con}, \mathsf{inv}) \end{aligned}$$

The supplier's interface V to the system is modeled in terms of the variables ship and pay. Constraints between these variables are used to characterize our requirements for the system. We want to ensure that the implementation upholds probity through this interface, that is, the combined constraints across $\mathsf{Impl} \equiv \mathsf{Clerk1} \otimes \mathsf{Clerk2} \otimes \mathsf{Appl}$ ensure $\mathsf{Probity}$:

$$(\mathsf{Impl})_{\Downarrow\{\mathsf{ship},\mathsf{pay}\}} \sqsubseteq \mathsf{Probity}$$

We are unconcerned about the possible values of the 'internal' variables inv and con and thus the constraint relation $\mathsf{Impl}_{\Downarrow\{\mathsf{ship},\mathsf{pay}\}}$ describes the constraints in Impl that exist between variables ship and pay. By definition, the above equation defines that all of the possible solutions of $\mathsf{Impl}_{\Downarrow\{\mathsf{ship},\mathsf{pay}\}}$ are solutions of $\mathsf{Probity}$, that is, for any assignment η of variables then

$$\mathsf{Impl}_{\Downarrow\{\mathsf{ship},\mathsf{pay}\}}\ \eta \leq_S \mathsf{Probity}\ \eta$$

The implementation must ensure that even in the presence of failures within the infrastructure (clerks), then it can still support the $\mathsf{Probity}$ requirement at it's external interface with the supplier. The behaviour of a failing (dishonest) clerk is not constrained by specifications $\mathsf{Clerk1}$ or $\mathsf{Clerk2}$. Thus, the constraints of the implementation where one clerk fails (constraint $true$) are ($\mathsf{Clerk2} \otimes \mathsf{Appl}$) or ($\mathsf{Clerk1} \otimes \mathsf{Appl}$) and we have

$$\begin{aligned} (\mathsf{Clerk2} \otimes \mathsf{Appl})_{\Downarrow\{\mathsf{ship},\mathsf{pay}\}} &\sqsubseteq \mathsf{Probity} \\ (\mathsf{Clerk1} \otimes \mathsf{Appl})_{\Downarrow\{\mathsf{ship},\mathsf{pay}\}} &\sqsubseteq \mathsf{Probity} \end{aligned}$$

This means that implementation can still ensure $\mathsf{Probity}$ even when a single clerk fails and we have verified that this particular separation of duty configuration is actually effective. As expected, the system is not resilient to the failure of both clerks:

$$\mathsf{Appl}_{\Downarrow\{\mathsf{ship},\mathsf{pay}\}} \not\sqsubseteq \mathsf{Probity}.$$

4 Quantitative Integrity Analysis

The previous section uses crisp constraints to describe system requirements and implementations. When a quantitative analysis of the system is required then it is necessary to represent these properties using soft constraints.

Consider again the $\mathsf{Probity}$ requirement and suppose that we aim not only to have a correct implementation, but, if possible, to have the "best" possible implementation. To do this we consider a soft constraint between variables ship and pay that assigns to the configuration $\mathsf{ship} = a$ and $\mathsf{pay} = b$ the preference level represented by the integer $a - b$[1]. If we are looking for the best implementation for the buyer, then we will try to maximize this level. In this way, different instances of the same system can be compared, and different implementations can be checked and analyzed.

Soft constraints also provide a basis for evaluating and comparing less resilient implementations that cannot uphold the intended requirement. For example, if an acceptable implementation Imp cannot be found to satisfy

$$\mathsf{Imp}_{\Downarrow\{\mathsf{ship},\mathsf{pay}\}} \sqsubseteq \mathsf{Probity}$$

then one might be satisfied (in some sense) by selecting the best of the less resilient, but acceptable implementations. Given insufficiently resilient implementations Imp_1 and Imp_2 then their corresponding semiring levels provide a relative ordering that allow the selection of the 'best' of the less resilient implementations.

Probabilistic based reasoning can also be done within the soft constraints framework. For example, consider an implementation $\mathsf{Imp3}$ that ensures that the number of payments is never more than 3, regardless of the number of shipments received. This is represented as:

$$\mathsf{Imp3} \equiv \mathsf{pay} \leq 3.$$

[1] This value represent how much pay differs from ship. Our goal is to have pay=ship, but sometimes this is impossible and our goal will be to minimize the $a - b$ difference.

Assume that there is a constraint on variable ship that specifies the probability of the possible number of shipments made at a certain time. If the nature of the probability distribution is such that it is generally more likely that the value of ship is greater than 3, then $\mathsf{Imp3}$ is a not unreasonable implementation (despite $\mathsf{Imp3}_{\Downarrow\{\mathsf{ship},\mathsf{pay}\}} \sqsubseteq \mathsf{Probity}$ not holding).

5 Discussion and Conclusion

By modeling the system and its infrastructure only in terms of abstract constraints, we argue that it becomes more realistic (than [9]) to consider modeling large complex application systems. Constraint solving can be done for large problems and, therefore, the proposed integrity analysis (as a constraint satisfaction problem) should scale up to similarly large/complex application systems configurations.

In [15] a policy analysis technique is proposed for detecting possible conflicts between separation of duty, user role assignment and role inheritance rules. This can be thought of as providing an analysis on, what is, in effect, the constraints on user role assignments. While useful, it is limited since it does not consider any further semantic information about the system and/or infrastructure. It would be interesting to apply the integrity analysis techniques proposed in this paper to extend the results in [15] for RBAC models.

Soft constraints can be used to provide a quantitative analysis of integrity. Section 4 outlines how soft constraints might be used to provide a measure of integrity to compare the effectiveness of different system configurations. If it is not possible to develop a resilient system configuration that fully meets the set of system requirements, then one may wish to consider the best of the less resilient, but acceptable (in some sense) configurations. For example, if it is not possible to configure a system that is resilient to all internal fraud, then an acceptable alternative might be to keep the fraud within some limit. Section 4 sketches a simplistic example of this; further research is required to develop this in general.

Another application of soft constraints to the analysis of integrity is that it allows the use of quantitative information in modeling the system and infrastructure configuration. By associating probability measures with component failures, we have used (elsewhere) soft constraints to validate that a system configuration/policy achieves integrity within some degree of probability.

In [1] soft constraints are used to represent confidentiality and authentication properties of security protocols. This approach is not unlike the strategy taken in this paper. The solution of the resulting constraint system gives a measure of the confidentiality/authentication of the system. In [1] a protocol run is compared with an "ideal" run without spies, When the solutions differ an attack to the protocol is identified. The proposed integrity analysis must consider various 'spy's, each characterizing the threats that a protection mechanism must withstand.

References

[1] G. Bella and S. Bistarelli. Soft constraints for security protocol analysis: Confidentiality. In *Proc. PADL'01*, volume 1990 of *Lecture Notes in Computer Science (LNCS)*. Springer, 2001.

[2] J. Bellone, A. Chamard, and C. Pradelles. Plane - an evolutive planning system for aircraft production. In *Proc. 1st Interantional Conference on Practical Applications of Prolog (PAP92)*, 1992.

[3] K. Biba. Integrity considerations for secure computer systems. Technical Report MTR-3153 Rev 1 (ESD-TR-76-372), MITRE Corp Bedford MA, 1976.

[4] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based Constraint Solving and Optimization. *Journal of the ACM*, 44(2):201–236, Mar 1997.

[5] S. Bistarelli, U. Montanari, and F. Rossi. Soft concurrent constraint programming. In *Proc. ESOP, April 6 - 14, 2002, Grenoble, France*, LNCS. Springer-Verlag, 2002.

[6] J. Bowen and D. Bahler. Constraint-based software for concurrent engineering. *IEEE Computer*, 26(1):66–68, Jan. 1993.

[7] D. D. Clark and D. R. Wilson. A comparison of commercial and military computer security models. In *Proceedings Symposium on Security and Privacy*, pages 184–194. IEEE Computer Society Press, Apr. 1987.

[8] Computer Security Institute/US Federal Bureau of Investigation. *Computer Crime and Security Survey*, 2001.

[9] S. Foley. A non-functional approach to system integrity. IEEE Journal on Selected Areas in Commications, 2003. *forthcoming*.

[10] T. Frühwirth and P. Brisset. Optimal planning of digital cordless telecommunication systems. In *Proc. PACT97*, London, UH, 1997.

[11] A. Konstantinou, Y. Yemini, S. Bhatt, and S. Rajagopalan. Managing security in dynamic networks. In *Proc. USENIX Lisa'99*, 1999.

[12] A. Mackworth. Constraint satisfaction. In S. Shapiro, editor, *Encyclopedia of AI (second edition)*, pages 285–293. John Wiley & Sons, 1992.

[13] J. Puget. A c++ implementation of clp. In *Proceedings of the 2nd Singapore International Conference on Intelligent Systems*, 1994.

[14] R. Sandhu et al. Role based access control models. *IEEE Computer*, 29(2), 1996.

[15] A. Schaad and D. Moffett. The incorportation of control principles into access control policies. In *Workshop on Policies for Distributed Systems and Networks*, Bristol, UK, 2001.

[16] U. S. Department of Defense. Integrity-oriented control objectives: Proposed revisions to the trusted computer system evaluation criteria (TCSEC). Technical Report DOD 5200.28-STD, U. S. National Computer Security Center, Oct. 1991.