Should We Prove Security Policies Correct?

Sebastiano Battiato Giampaolo Bella Salvatore Riccobene

Dipartimento di Matematica e Informatica, Università di Catania
Viale A.Doria, 6 — I-95125 Catania, ITALY

Abstract. Security policies are abstract descriptions of how a system *should* behave to be secure. They typically express what is obligatory, permitted, or forbidden in the system. When the system is implemented, its formal verification consists in checking whether it conforms to the norms that its policy stated. Hence, security policies significantly influence the final assessment of real systems. Experience shows that important policies suffering inconsistencies have reached the final stage of implementation in a real system. Here comes the need for formal analysis at the abstract level of policies. It is advocated that known inductive techniques and a general-purpose proof assistant can be used profitably for the proof of correctness of security policies.

1 Introduction

A *security policy* for a generic system can be informally explained as a set of norms (rules) stating who can or cannot do what in the system. Security policies underlie all areas of Computer Science where some form of security is required, ranging from databases to operating systems, but also pervade the real world. For example, "access to record i ought to be forbidden at week-ends" might be part of the security policy for a DBMS. Likewise, "in case of fire, one ought to leave the building" is (part of) the security policy for modern constructions. The examples may raise the intuition that security policies must enjoy some *correctness* property.

The motivations for our work are at least twofold. A policy does not describe how a system behaves or how it is perceived to behave, but, rather, how it *should* behave to be *secure*, whatever the adjective *secure* means for the system. That ideal behaviour is exactly what one needs to (and must!) keep in mind when one is verifying a real system. The system passes the verification when it is shown to conform to its intended functioning specified in its security policy. It follows that the security policy plays a major role in the final assessment of the real system. But recent experience tells that the huge policy underlying a huge e-commerce protocol like SET may suffer significant incompleteness (omitted features) or ambiguity (unclear features) [1, 2]. We advocate that a method for formally verifying abstract system properties at the policy level is needed.

Research efforts in the analysis of security policies already exist, but they appear to have succeeded mostly in terms of specification rather than of verification. A variety of modal logics are used as specification languages, equipped with three *deontic* operators, Obligatory, Permitted and Forbidden, which formalize the respective *modalities*:

Battiato S., Bella G. and Riccobene S. (2004).
Should We Prove Security Policies Correct?.
In *Proceedings of the 1st International Workshop on Electronic Government and Commerce: Design, Modeling, Analysis and Security*, pages 56-65
DOI: 10.5220/0001389600560065

obligations, permissions and interdictions. Deontic logics are typically endowed with systems of inference rules whereby given facts of the logics can be proved or refuted [3, 4]. To our current understanding, what appears to be missing is some mechanism for the automatic generation of tests, hence a full verification method. For example, let us consider a security policy for file-system access. Checking that a candidate norm such as "a System Security Officer is permitted to write on system files" induces no contradiction is indeed interesting. But verifying that none of the norms for a System Security Officer induces contradiction or that no norm at all induces contradiction would be deeper results, much closer to some notion of *policy correctness*. Equivalent terminology is *policy consistency*.

Absence of *contradictions* and absence of *dilemmas* are the basic requirements of policy correctness. There is a contradiction when something is at the same time permitted and forbidden, or at the same time obligatory and forbidden — for example, "Bob is permitted to issue a nonce; Bob is forbidden to issue a nonce". There is a dilemma when both something and its opposite are obligatory, or both are forbidden — for example , "it is obligatory that Alice registers her public key; it is obligatory that Alice does not register her public key". The specific application domain may impose additional requirements to policy correctness, such as *secrecy*, i.e. protecting resources from agents with insufficient rights, and *fairness*, i.e. distributing resources equally

Our idea is to use mathematical induction to model security policies. We borrow concepts from the inductive method to verifying security protocols introduced by Paulson [5] and developed with Bella [6, 7]. Each policy norm can be expressed as an inductive rule extending a given trace of norms. The security policy can then be modelled inductively as the set of all possible traces of norms that the policy admits. Agents and actions to which the norms apply can be specified for the sake of expressiveness, or left as generic members of unbounded sets. Once the policy is a set built up by induction, the corresponding inductive principle can be used to prove safety properties of the set. The basic requirements of policy correctness indeed are safety properties. This position paper sets up and demonstrates the foundations of our approach to verifying security policies, while the next step is to complete mechanizing the approach with the proof assistant Isabelle/HOL [8]. The language of Higher Order Logic will make the specification of policy properties easier, while Isabelle's simplifier will resolve the trivial cases without human intervention. These features will make the analysis of vast policies possible. If we compare security protocols and security policies in terms of specification efforts, we expect many more, though simpler, rules in the case of policies. Hence, the human efforts required by the verification can be expected to be smaller.

The structure of this position paper is simple. The treatment of security policies (§2) begins with the concepts of contradictions and dilemmas and terminates with a published, example policy [9]. Our inductive modelling of policies and their properties comes next (§3). A few remarks conclude (§4).

2 Security policies

A security policy explains how a system should behave to be considered secure. To take an example from the real world, modern building evacuation systems are considered

secure when the people in the building do follow the security policy to evacuate in case of fire — people are part of the system. The law inspectors will simply sign a conformity declaration for the building if it has got a reasonable security policy and its inhabitants are informed of it. This confirms once more that it is impossible to claim security for a system that is missing a security policy.

At this stage, we can define a security policy more formally (Definition 1).

Definition 1 (Security Policy). *A* security policy *is a set of norms regulating the modalities — obligations, permissions, interdictions — for a set of agents on some actions.*

It is worth remarking that a security policy may be a very large set of norms. Consider the policy regulating the internal functioning of a financial institution, for example. Hundred, if not thousand norms must coexist together, and if it is easy and quick to state a single, meaningful norm, it is hard to derive a full understanding of how the new norm will impact the existing kernel [10].

2.1 Modalities

The following treatment develops on a generic action a, which is formalized as a boolean. This makes it possible to logically relate actions using conventional boolean connectives.

As mentioned above, the basic requirements towards correctness of a policy are absence of contradictions and absence of dilemmas. Studying these requirements imposes a careful understanding of the semantics of the modalities. Obligatory is the *basic modality*, in the sense that all other modalities are defined from it as follows.

Definition 2 (Permission). *An action is* permitted *if and only if not performing it is not obligatory:*

$$\mathsf{Permitted}(a) \triangleq \neg\mathsf{Obligatory}(\neg a)$$

Definition 3 (Interdiction). *An action is* forbidden *if and only if not performing it is obligatory:*

$$\mathsf{Forbidden}(a) \triangleq \mathsf{Obligatory}(\neg a)$$

Security policies are sometimes enriched with another modality, Waived, expressing that an action must not necessarily done. This also can be defined in terms of the basic modality as its logical negation.

Definition 4 (Waiver). *An action is* waived *if and only if it is not obligatory to perform it:*

$$\mathsf{Waived}(a) \triangleq \neg\mathsf{Obligatory}(a)$$

2.2 Contradictions

If an action is permitted, then having it forbidden too is a contradiction according to common understanding. (What if you read the following public notice: smoking is forbidden, and smoking is permitted!). An even more trivial contradiction would be to

have an action waived and at the same time obligatory. These may be formalized by dedicated definitions, but it is more important to investigate what the general structure of a contradiction is. It would make it conceivable to check an entire policy against that structure, so as to verify whether contradictions exist at all.

2.3 Dilemmas

If it is forbidden to perform an action and at the same time not to perform it, this clearly is a dilemma, because it would make it impossible not to violate the policy. (What if you read the following public notice: smoking is forbidden, and not smoking is forbidden!). However, it is not a dilemma that it is waived to perform an action and at the same time not to perform it, because such a combination just establishes freedom upon that action. Also dilemmas can be easily formalized by dedicated definitions, though a general version of their structure would be more important towards verifying policy correctness.

2.4 An example policy

As an example, we introduce in Figure 1 a simple, published policy for file system access due to Cholvy and Cuppens [9], slightly amended by Bella [10]. Each modality is abbreviated by its initial. The users of the file system can play four roles. Norms N1 to N4 state that a normal user (who plays role user) is permitted to read any public file, but to write only on those he owns. He is also forbidden to downgrade any files, and obliged to change his password if it is old. Norms N5 and N6 state that a secret user (who plays role secret) is also permitted to read or write on secret (i.e. non-public) files. Norm N7 states that a system security officer (who plays role sso) is permitted to downgrade any file. The last norm is for bad users (who play role bad), those who have for example forgotten to change their password for too long. The norm forbids all actions to bad users.

Each policy norm is simple, but the policy as a whole is complex because of the obvious constraints on roles. Secret users are in fact also users; system security officers are also secret users; bad users may be either users or secret users. These constraints should also be composed using a transitivity relation, yielding for example that system security officers also are users. With these constraints, it is not trivial to get a synergic understanding of the entire policy. Just to anticipate a contradiction, notice that norms N3 and N7 cause a contradiction on action $\mathsf{downgrade}(a, f)$ because a system security officer is also a normal user. A more systematical study on whether the policy suffers other problems is presented later (§3).

3 An inductive approach to policy verification

Our idea is to formalize a security policy as the set of all admissible traces of norms. A trace of norms can be interpreted as a possible reduction of the policy to specific actions. It is the same idea that turned out successful with security protocols [6, 5]. The

N1: if play(a,user) and public(f) then P(read(a,f))	N2: if play(a,user) and public(f) and owner(f,a) then P(write(a,f))
N3: if play(a,user) then F(downgrade(a,f))	N4: if play(a,user) and password(a,p) and old(p) then O(change_password(a))
N5: if play(a,secret) and not(public(f)) then P(read(a,f))	N6: if play(a,secret) and not(public(f)) and owner(f,a) then P(write(a,f))
N7: if play(a,sso) then P(downgrade(a,f))	N8: if play(a,bad) then F(read(a,f)) F(write(a,f)) F(downgrade(a,f)) F(change_password(a))

Fig. 1. An example security policy for file-system access [9]

set is defined by induction, and properties of interest can be proved about it using the corresponding induction principle.

As in the previous section, a generic action is formalized here as the boolean variable a. In consequence, asserting a indicates that the corresponding action is performed, while asserting $\neg a$ indicates that the same action is not performed.

3.1 Modelling a policy

If $\mathcal{P}$ is declared as a set of traces of norms, it can be defined inductively as follows.

- The empty trace belongs to $\mathcal{P}$.
- Extending a trace of $\mathcal{P}$ with a norm stated by the policy yields a trace of $\mathcal{P}$.

The example policy in Figure 1 can be formalized accordingly, as in Figure 2. Fat square braces denote rule preconditions, while $\#$ is the list cons operator.

Here, agents and files have been formalized as freetypes, and agent roles as a datatype. The constraints on roles can be trivially introduced as axioms of implicational form, such as $\mathsf{play}(a, \mathsf{secret}) \rightarrow \mathsf{play}(a, \mathsf{user})$. The modalities can be defined in terms of the basic modality, according to definitions 2, 3 and, if needed, 4.

A remark is necessary about *consequential policies*, which apply some norms on the basis that other norms apply. For example, we may have a policy rule stating that if a secret user is permitted to read secret files, then so is a system security officer. These can be modelled inductively without extra efforts. Our example policy is not consequential.

Base: [] $\in \mathcal{P}$

N1: [| nms1 $\in \mathcal{P}$; play(a,user); public(f) |]
$\Longrightarrow$ P(read(a,f)) # nms1 $\in \mathcal{P}$

N2: [| nms2 $\in \mathcal{P}$; play(a,user); public(f); owner(f) |]
$\Longrightarrow$ P(write(a,f)) # nms2 $\in \mathcal{P}$

N3: [| nms3 $\in \mathcal{P}$; play(a,user) |]
$\Longrightarrow$ F(downgrade(a,f)) # nms3 $\in \mathcal{P}$

N4: [| nms4 $\in \mathcal{P}$; play(a,user); password(a,p); old(p) |]
$\Longrightarrow$ O(change_password(a)) # nms4 $\in \mathcal{P}$

N5: [| nms5 $\in \mathcal{P}$; play(a,secret); ¬public(f) |]
$\Longrightarrow$ P(read(a,f)) # nms5 $\in \mathcal{P}$

N6: [| nms6 $\in \mathcal{P}$; play(a,secret); ¬public(f); owner(f,a) |]
$\Longrightarrow$ P(write(a,f)) # nms6 $\in \mathcal{P}$

N7: [| nms7 $\in \mathcal{P}$; play(a,sso) |]
$\Longrightarrow$ P(downgrade(a,f)) # nms7 $\in \mathcal{P}$

N8: [| nms8 $\in \mathcal{P}$; play(a,bad) |]
$\Longrightarrow$ F(read(a,f)) # F(write(a,f) #
F(downgrade(a,f) # F(change_password(a)) # nms8
$\in \mathcal{P}$

Fig. 2. Modelling the example policy inductively

3.2 Modelling and proving policy properties: sanity checks

A crucial issue in formal verification is to build a realistic model. As a form of sanity check, we can prove a trivial, subsidiary lemma.

Lemma 1 (Permission w.r.t Interdiction). *If an action is permitted, then it is not forbidden:*

$$\mathsf{Permitted}(a) \rightarrow \neg\mathsf{Forbidden}(a)$$

Proof. By definitions 2 and 3.

When the proof is conducted on a theorem prover, the definitions are installed as simplification rules, hence a single appeal to the simplifier solves the theorem. It may be interesting to look at the converse lemma.

Lemma 2 (Interdiction w.r.t Permission). *If an action is forbidden, then it is not permitted:*

$$\mathsf{Forbidden}(a) \rightarrow \neg\mathsf{Permitted}(a)$$

Proof. Converse of Lemma 1. Alternatively, by definitions 2 and 3.

3.3 Modelling and proving policy properties: no contradictions

As we are working towards a mechanization on a proof assistant, it is preferable to define the policy properties in their simplest form. All definitions can be used as simplification rules to prove more complex properties, perhaps not of immediate understanding, by reducing them to the simple ones.

As for contradictions, we have come to the conclusion that their general structure can be expressed as in the following definition.

Definition 5 (Contradiction). *An action leads to* contradiction *if and only if it is at the same time obligatory and not obligatory, or not to perform it is at the same time obligatory and not obligatory:*

$$\begin{aligned} \mathit{contradiction}(a) \triangleq\ & (\mathsf{Obligatory}(a) \wedge \neg\mathsf{Obligatory}(a)) \vee \\ & (\mathsf{Obligatory}(\neg a) \wedge \neg\mathsf{Obligatory}(\neg a)) \end{aligned}$$

Our definition should be read as the simplest form of contradiction, although its intuitiveness may be subject to debate. Other, more elaborated, contradictions (such as those mentioned above, §2.2) can be simplified into this, as expressed by the following theorems.

Theorem 1. *If an action is both permitted and forbidden, then it leads to contradiction:*

$$\mathsf{Permitted}(a) \wedge \mathsf{Forbidden}(a) \rightarrow \mathit{contradiction}(a)$$

Proof. By Lemma 1 and definitions 3 and 5. Alternatively, by Lemma 2 and definitions 2 and 5. Alternatively, by definitions 2, 3 and 5.

Theorem 2. *If an action is both waived and obligatory, then it leads to contradiction:*

$$\mathsf{Waived}(a) \wedge \mathsf{Obligatory}(a) \rightarrow \mathit{contradiction}(a)$$

Proof. By definitions 4 and 5.

3.4 Modelling and proving policy properties: no dilemmas

We have also defined the simplest structure for dilemmas as in the following definition.

Definition 6 (Dilemma). *An action leads to* dilemma *if and only if it is at the same time obligatory to perform it and not to perform it:*

$$\mathit{dilemma}(a) \triangleq \mathsf{Obligatory}(a) \wedge \mathsf{Obligatory}(\neg a)$$

Given the simplest form of a dilemma, more elaborated situations in fact are dilemmas if they can be reduced to that form. The following theorem is elementary to prove but significant. Theorems of the same form can be analogously proved also for the other two modalities.

Theorem 3. *If an action is forbidden and so is not to perform it, then it leads to dilemma:*

$$\mathsf{Forbidden}(a) \wedge \mathsf{Forbidden}(\neg a) \rightarrow dilemma(a)$$

Proof. By definitions 3 and 6.

The notions introduced so far are intuitive but may be tricky. For example, the reader may notice that Definition 6 is missing some symmetry with respect to Definition 5. In particular, one may be tempted to enrich it with a second disjunct and obtain

$$(\mathsf{Obligatory}(a) \wedge \mathsf{Obligatory}(\neg a)) \ \vee \ (\neg\mathsf{Obligatory}(a) \wedge \neg\mathsf{Obligatory}(\neg a))$$

However, the second disjunct is logically equivalent, by Definition 4, to

$$\mathsf{Waived}(a) \wedge \mathsf{Waived}(\neg a)$$

which clearly must not count as a dilemma because if an action is waived and so is not to perform it, then there just is augmented freedom.

3.5 Modelling and proving policy properties: correctness

Absence of contradictions and dilemmas is the basic requirement of policy correctness (plus others, depending on the application domain).

If the policy is modelled as an inductively defined set, we would like to establish those properties on every trace of the set. To do so, we must lift our definitions of contradiction and dilemma over traces. Below, set turns a list into the set of elements of the list.

Definition 7 (Contradiction on Trace). *An action leads to* contradiction on a trace *if and only if there exists an action that leads to contradiction by means of norms of the trace:*

$$\begin{aligned} contradiction(a,\ nms) \triangleq\ & (\mathsf{Obligatory}(a) \in \mathsf{set}\ nms \wedge \neg\mathsf{Obligatory}(a)) \in \mathsf{set}\ nms \vee \\ & (\mathsf{Obligatory}(\neg a) \in \mathsf{set}\ nms \wedge \neg\mathsf{Obligatory}(\neg a) \in \mathsf{set}\ nms) \end{aligned}$$

Definition 8 (Dilemma on Trace). *An action leads to* dilemma on a trace *if and only if there exists an action that leads to dilemma by means of norms of the trace:*

$$dilemma(a,\ nms) \triangleq \mathsf{Obligatory}(a) \in \mathsf{set}\ nms \wedge \mathsf{Obligatory}(\neg a) \in \mathsf{set}\ nms$$

Definition 9 (Policy Correctness). Correctness *of a security policy holds if and only if its inductive model $\mathcal{P}$ does not suffer any contradictions or dilemmas for any actions on any of its traces:*

$$\begin{aligned} correctness(\mathcal{P}) \triangleq\ & \forall\ nms\ a. \\ & nms \in \mathcal{P} \rightarrow \neg contradiction(a,\ nms) \wedge \neg dilemma(a,\ nms) \end{aligned}$$

If we attempt to prove this property with the help of a theorem prover, should the property fail, the prover would exhibit the counterexample showing which action and which norm contradict the property on which trace.

3.6 On the example policy

If we attempt to prove correctness of the example policy, the inductive proof will fail at several stages. For example, in case of norm N7, proceeding backward the prover simplifies $\neg\mathsf{Permitted}(\mathsf{downgrade}(a, f))$ to $\mathsf{Obligatory}(\neg\mathsf{downgrade}(a, f))$. We need to contradict the latter fact to close the proof. A dedicated subproof would show that it is impossible in case a also is a normal user, due to norm N3. This is more entangled to describe than to verify even by pen-and-paper, let alone with the help of a proof assistant. The constraints of the system on agent roles must be stated because the prover will ask us to address the case when agent a of norm N7 matches agent a of norm N3. The constraints tell that they can match, hence a contradiction on the corresponding actions is possible.

The example policy hides a number of other contradictions (not of interest in this paper) but no dilemma. For the sake of demonstration, we can imagine to extend the policy with an intuitive rule that waives system security officers from changing their password when it is old. The policy model can easily account for the change by including the extra rule given in Figure 3.

```
N9: [| nms9 ∈ P; play(a,sso); password(a,p); old(p) |]
    ⟹ W(change_password(a)) # nms9 ∈ P
```

Fig. 3. An extra policy rule leading to dilemma

A putative proof that the policy does not suffer dilemmas (towards a proof of policy correctness) can be conducted by inductive arguments similar to those given above. It would fail at least on the trace of norms where both norms N4 and N9 applied, due to the constraints on agent roles.

4 Conclusions

Security policies are typically analyzed using dedicated modal logics. The existing literature supports the claim that verification in this field lacks mechanization. It is important to verify using the calculus of the logic that a given candidate leads to contradiction or dilemma. It would be more important to either verify that there is no such candidate or to exhibit such a candidate.

This can be done using an inductive method to modelling and verifying policies. We intend to mechanize the method with the help of a proof assistant such as Isabelle. The prover will hold the burden of simplifying expressions, thus making it possible to cope with large policies. Having done that, testing the method on real-world policies will be possible.

We have provided inductive definitions for absence of contradictions and absence of dilemmas, the basic components of protocol correctness. Other components can be added but the foundational treatment is already available at present.

References

1. Bella, G., Massacci, F., Paulson, L.C.: Verifying the SET Registration Protocols. IEEE Journal of Selected Areas in Communications **21** (2003) 77–87
2. Bella, G., Massacci, F., Paulson, L.C., Tramontano, P.: Formal Verification of Cardholder Registration in SET. In Cuppens, F., Deswarte, Y., Gollmann, D., Waidner, M., eds.: Proc. of the 6th European Symposium on Research in Computer Security (ESORICS 2000). LNCS 1895, Springer-Verlag (2000) 159–174
3. Cholvy, L.: Checking Regulation Consistency by using SOL-resolution. In: International Conference on Artificial Intelligence and Law. (1999) 73–79
4. Cuppens, F., Saurel, C.: Specifying a Security Policy: A Case Study. In: Proc. of the 9th IEEE Computer Security Foundations Workshop, IEEE Press (1996)
5. Paulson, L.C.: The Inductive Approach to Verifying Cryptographic Protocols. Journal of Computer Security **6** (1998) 85–128
6. Bella, G.: Inductive Verification of Smart Card Protocols. Journal of Computer Security **11** (2003) 87–132
7. Bella, G., Paulson, L.C.: Mechanical Proofs about a Non-Repudiaton Protocol. In Boulton, R.J., Jackson, P.B., eds.: Proc. of the 14th International Conference on Theorem Proving in Higher Order Logics (TPHOLs'01). LNCS 2152, Springer-Verlag (2001) 91–104
8. Paulson, L.C.: Isabelle: A Generic Theorem Prover. LNCS 828. Springer-Verlag (1994)
9. Cholvy, L., Cuppens, F.: Analyzing Consistency of Security Policies. In: Proc. of the 16th IEEE Symposium on Security and Privacy, IEEE Press (1997)
10. Bella, G.: Interactive Simulation of Security Policies. In Panda, B., ed.: 17th ACM Symposium on Applied Computing, ACM Press and Addison Wesley (2002) 242–252