A Declarative Model for Reasoning about Form Security

Aaron Hunter
British Columbia Institute of Technology, Burnaby, Canada

Keywords: Reasoning about Action, Information Security, Knowledge Representation.

Abstract: We introduce a formal methodology for analysing the security of digital forms, by representing form signing procedures in a declarative action formalism. In practice, digital forms are represented as XML documents and the security of information is guaranteed through the use of digital signatures. However, the security of a form can be compromised in many different ways. For example, an honest agent might be convinced to make a commitment that they do not wish to make or they may be fooled into believing that another agent has committed to something when they have not. In many cases, these attacks do not require an intruder to break any form of encryption or digital signature; instead, the intruder simply needs to manipulate the way signatures are applied and forms are passed between agents. In this paper, we demonstrate that form signing procedures can actually be seen as a variation of the message passing systems used in connection with cryptographic protocols. We start with an existing declarative model for reasoning about cryptographic protocols in the Situation Calculus, and we show how it can be extended to identify security issues related to digital signatures, and form signing procedures. We suggest that our results could be used to help users create secure digital forms, using tools such as IBM's Lotus Forms software.

1 INTRODUCTION

Information is often exchanged over a network through the use of *digital forms*. A form is essentially a structured collection of data that allows an agent to enter information, and also to commit to particular claims through the use of a signature. In the past, the security of a signature was guaranteed through the physical properties of paper. In the case of digital forms, security must be demonstrated through some form of mathematical proof. It is well known that multi-agent systems with formal models of knowledge can be used to prove the security of cryptographic protocols, but such methods have not been widely employed in the context of form validation. In this paper, not only do we demonstrate that the same methods can be used to prove the security of digital forms, but we also show that it is possible to simply modify some existing, declarative protocol verification formalisms for use with digital forms. The result is a declarative model of form structure that can be used to authomatically find attacks on forms.

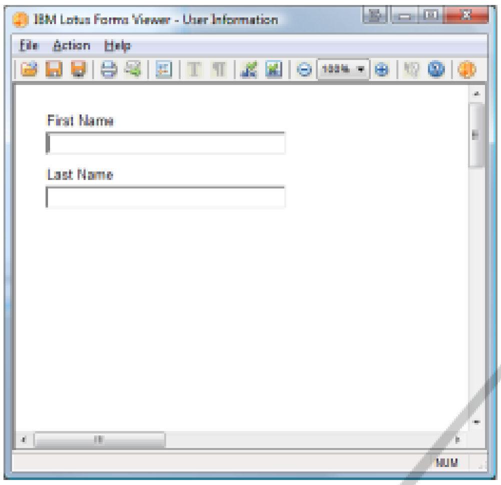
Our framework is based on a Situation Calculus (SitCalc) model of a message passing system, in which honest agents pass simple messages on an insecure channel. In the original framework, messages are essentially text and all attacks are based on intercepting or mirroring messages. This paper makes two contributions to existing work on logic-based models for reasoning about security. First, we provide the foundation for an executable model of digital forms that allows attacks on forms to be automatically discovered. This is a problem of significant practical interest that has not been addressed substantially in existing work. Second, our work further demonstrates the utility of declarative models for reasoning about security. Since the original model for protocol verification is declarative, it is actually quite straightforward to modify the solver to deal with a different security domain.

2 PRELIMINARIES

2.1 Motivation

It is well known that businesses stand to save a great deal of money by transitioning from paper forms to digital forms (Bertrand et al., 1995). However, while HTML forms on the Internet are very common, they are known to suffer from many limitations both in expressiveness and security. For example, paper forms

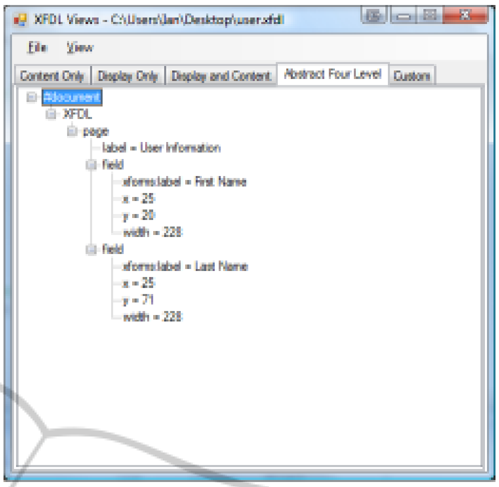
Hunter A..
A Declarative Model for Reasoning about Form Security.
DOI: 10.5220/0005213604200425
In *Proceedings of the International Conference on Agents and Artificial Intelligence* (ICAART-2015), pages 420-425
ISBN: 978-989-758-074-1



First Name

Last Name

Figure 1: A Simple Form.



```
#document
  XFDL
    page
      label = User Information
      field
        xforms:label = First Name
        x = 25
        y = 20
        width = 228
      field
        xforms:label = Last Name
        x = 25
        y = 71
        width = 228
```

Figure 2: The Corresponding XML Tree.

typically allow an agent to sign just a part of the form in sequence with other agents. Moreover, as indicated previously, the data on a paper form is protected by the physical properties of paper that allow tampering to be predicted. As a result, while businesses would prefer to use digital forms, they would like forms that preserve the standard conventions and capabilities of paper forms. This has lead to the development of specialized form processing software.

The standard approach is to use XML standards to represent the structure of a form. For example, the XFDL standard is used by some form processing software (Boyer, 2005). XML trees in this standard use nodes to represent form fields, and the children of each node represent properties of the corresponding field. Field properties might include location, text, font size, or any number of other features. As an illustration, Figure 1 provides the standard view of a simple form. Figure 2 provides a view of the XML tree for the XFDL representation of this form.

2.2 Logical Representations of Forms

There has been some past work on formal representations of XML forms. In particular, a precise definition of a *document tree* is introduced in (Hunter, 2012). The idea is to identify a form with a tree that is roughly the same shape as the corresponding XFDL representation. A vocabulary is introduced consisting of *labels* and *values*, and then a specialized property is introduced to indicate if a form field is visible or not. For example, the set of labels might include the word *name* and the set of values might include all names of relevant agents. In practice, both sets would actually be generated from a finite alphabet of symbols. Given the set of labels and values, one can define a propositional vocubulary consisting of label-value pairs. Hence, when a form is filled out, the author is actually asserting that certain labels have certain values. Using this basic framework, the *content* of a form is then defined to be a formula over the defined propositional vocabulary. Modal operators for belief can then be introduced for each agent, which are then used to define the notion of commitment. This notion can then be used to define attacks in terms of forcing *false commitments* or other forms of *deception*.

The preceding formal model of digital forms helps to make some attacks explicit, but it suffers from two main deficiencies. First of all, the approach is not declarative. In order to represent a form or an attack on a form, one needs to delve into the internal representation in terms of trees. This makes it somewhat difficult to extend and modify the approach for different kinds of documents. The second problem is that this is not a model that lends itself to efficient automated reasoning. After creating the logical representation of a form, the only way to find attacks on the form is through exploration by hand.

2.3 Cryptographic Protocol Verification

Logical approaches to the verification of cryptographic protocols originated with the pioneering work on BAN logic (Burrows et al., 1990). The basic idea of BAN logic was to use a logic to model and reason about the beliefs of agents participating in the cryptographic protocol. The logic itself was not particularly sophisticated, so it is not longer considered a viable approach to protocol verification. However, the notion of using a logical model that includes the beliefs of agents was very influential. Many related approaches have followed, including representations based on epistemic logics (van der Hoek and Wooldridge, 2002), multi-agent systems (Fagin et al., 1995), strand spaces (Halpern and Pucella, 2003), and logic programs (Carlucci Aiello and Massacci, 2001). In each case, the intruder model used is called the Dolev-Yao intruder (Dolev and Yao, 1983). In this model, the intruder is able to intercept messages, redirect messages, and spoof messages. The main limitation of this intruder is that they are not able to decipher encrypted messages; this is a reasonable limitation if we assume strong encryption with large key sizes.

2.4 The Situation Calculus

For the present paper, we are interested in declarative models for reasoning about cryptographic protocols, with particular emphasis on the SitCalc. In the interest of space, we do not introduce the SitCalc here; we refer the reader to (Levesque et al., 1998) for a basic introduction. For now we simply introduce the basic terminology. In the SitCalc, there are several important categories of entities. First, there are *situations* (denoted by the variable s). A situation represents a configuration of the world, along with the history of all actions that have been executed. There are also *actions* (denoted by the variable A), that represent things that an agent can do to change the state of the world. There is also a function symbol do, where $do(s,A)$ is understood to represent the situation that results when the action A is executed in the situation s. There is a distinguished constant s_0 representing the initial situation, so that every situation can be represented as $do(\bar{A},s_))$ for some sequence of actions $\bar{A}$. A single second-order induction axiom allows us to prove that certain properties hold for all situations.

In the SitCalc, a predicate that takes a situation as an argument is called a *fluent*. Informally, fluents are used to represent properties of the world that may change due to the effects of actions. For example, in a message passing system, there could be a fluent *holds* that indicates if a certain agent is holding a particular message in a particular situation. Action domains are fomalised in the SitCalc by explicitly specifying the way that fluent values change when actions are executred. The effects of actions are given by specifying precondition and effect axioms that indicate which predicate values change in a situation when an action is executed. One of the advantages of the SitCalc for reasoning about actions is that SitCalc formilizations can be translated into GoLog, an executable logic-based language for planning.

The SitCalc formalization of message passing in (Hunter et al., 2013) includes four different kinds of entities: *Agents*, *Nonces*, *Keys* and *Data*. A *nonce* is a random number used in a cryptographic protocol. The main actions in the formalization are *makeNonce*, *encrypt*, *decrypt*, *compose*, *send* and *receive*. Using these basic entities and actions, a set of axioms is introduced that specifies the effects of actions and the capabilities of the intruder. This SitCalc formulation is then translated into a GoLog program that can be executed to automatically find attacks on the protocol in a reasonable time frame. Our approach in this paper is to extend this model for validating security properties on digitally signed forms.

3 FORMS AS PROTOCOLS

3.1 Intuition

In this section, we demonstrate how to develop a declarative treatment of forms by treating forms as a protocols. The first observation to make is that signing a form is actually a multi-step procedure that involves the exchange of information. The basic steps are as follows:

1. An agent A creates a form, requiring n of signatures (possibly on different parts of the form).
2. Repeat until all unsigned parts have been signed:
   (a) The agent holding the form signs any number of empty parts.
   (b) The agent passes the form to some other agent to hold.
3. The completed form is passed to some authority.

We will refer to this process as a *form signing procedure*.

Our basic approach is to express form signing procedures as a kind of message passing protocol. There are several steps required:

- Replace messages with *forms*.
- Introduce actions for filling and signing forms.
- Specify the exact steps to a signing procedure, *as a protocol*.

By formalizing all of these steps, we can automatically find some forms of attack on a form signing procedure.

3.2 Formalization

Note that there are already pre-existing SitCalc formalizations of message passing, so we do not want to replicate the work involved in developing such systems. Two representative SitCalc formalizations are

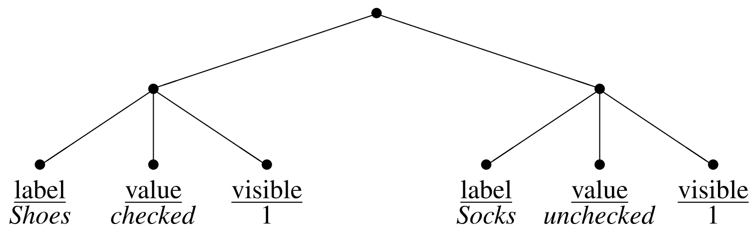


Figure 3: A Simple Form Tree.

given in (Hernández-Orallo and Pinto, 2000)) and (Hunter et al., 2013), respectively. We use the latter as a starting point for this work, as it includes a translation to GoLog for authomatically finding attacks. Our approach is to extend this SitCalc theory by introducing *forms* that can be sent and received. Since we are assuming an underlying SitCalc theory for message passing, we are also assuming some of the utility actions required for constructing and sending messages. In particular, it is useful to assume an underlying action *list* that takes a sequence of n objects and produces a single object representing the list. In the remainder of this section, we describe the main categories of object required for our formalization.

Items. The variable i, possibly with subscripts, will range over form *items*. The intention is that an item represents some relevant component of a form, such as an input mechanism or a text label. For the sake of simplicity, we think of all inputs as simple text blocks or as text blocks with an associated checkbox for input.

We introduce a functional fluent $newinput(p,s)$ that is used to generate a new input to place on a form, labelled with the proposition p. The following fluents apply to inputs:

- $Label(i,p,s)$ – True just in case the item is labelled with the proposition p.
- $Checked(i,s)$ – True just in case the associated checkbox has been checked.
- $Invisible(i,s)$ – True just in case the item is invisible.

In practice, there are many additional options that can be associated with a given item. For example, every item typically has some associated coordinates indicating a position on the page. The simple set provided here is sufficient for an initial formal analysis.

Example. Suppose we would like a form to collect data about the clothes people are wearing. We might decide one item needes would be a checkbox labelled "Shoes". The process here would be to create an empty form, then create an item with the label shoes and add it to the form. We could then create another item labelled "Socks" and add it to the form. The XFDL tree representing this form is in Figure 3. For now, every form that we consider has this simple structure; but it would be straightforward to produce more complex forms by adding new kinds of items and new kinds of properties.

Forms. The primary activity in our framework is the manipulation and exchange of forms. The variable f, possibly with subscripts, will range over *forms*. A form is primarily a collection of inputs that can be signed. In addition to the unary predicate *Form*, we introduce a fluent symbol $Contains(f,x,s)$ that is true just in case the form f contains in the input x in situation s. One important feature that must be captured is the fact that signatures do not always apply to an entire form. For this, we need to introduce some formal mechanism for identifying a sub-form to be signed. In order to reduce the number of object types required, we handle this problem by introducing the fluent $SubForm(f_1,f_2,s)$ which is true just in case every item in f_1 is also in f_2, in the situation s.

Sendables. The introduction of forms into the vocabulary results in some potential redundancy when it comes to sending and receiving information. In particular, both messages and forms can now be transmitted between agents in (essentially) the same manner. As such, we introduce a new type called *sendable*, which is basically an abstract class that includes both messages and forms.

3.3 Action Effect Axioms

We require actions for modifying and signing forms. In the SitCalc, each action requires a precondition and an effect. We remark that these conditions use a distinguished predicate called *designer* to indicate if a given agent is able to design forms. The following actions are required:

1. $sign(a,f_s,f,k)$ – Agent a signs the subform f_s of f using key k.

Precondition:
$Poss(sign(a,f_s,f,k),s) \equiv (Has(a,f,s) \wedge (Has(a,k,s) \vee \exists a' PublicKey(a',k)))$
Effect:
$Has(a,signed(f,k),do(sign(a,m,k),s))$

2. $check(a,i)$ – Agent a checks the item i.
Precondition:
$Poss(check(a,i,s)) \equiv Has(a,i,s) \wedge \neg checked(i,s)$
Effect:
$checked(i,do(check(a,i,s)))$
3. $uncheck(a,i)$ – Agent a checks the item i.
Precondition:
$Poss(uncheck(a,i,s)) \equiv Has(a,i,s) \wedge checked(i,s)$
Effect:
$\neg checked(i,do(check(a,i,s)))$
4. $send(a_1,a_2,f)$ – Agent a_1 sends the form to a_2. An intruder can masquerade as a_2.
Precondition:
$Poss(send(a_1,a_2,f),s) \equiv ((Has(a_1,f,s) \wedge a_1 \neq a_2) \vee Has(intr,f,s))$
Effect:
$Sent(f,a_1,a_2,do(send(a_1,a_2,f),s))$
5. $receive(a_1,a_2,f)$ – Agent a_1 receives the form f from a_2. An intruder can intercept messages.
Precondition:
$Poss(receive(a_1,a_2,f),s) \equiv (Sent(f,a_2,a_1,s) \vee (a_1 = intr \wedge \exists a' Sent(f,a_2,a',s)))$
Effect:
$Has(a,f,do(receive(a_1,a_2,f),s)) \wedge \neg Sent(a_2,a_1,f,do(receive(a_1,a_2,f),s))$
6. $composeform(a,f,\bar{i})$ – Agent a composes form f consisting of items $\bar{i} = i_1,\ldots,i_n$. Consequently, $composeform$ is in fact a set of actions, one for each possible number of arguments.
Precondition:
$Poss(composeform(a,f,list(\bar{i}),s) \equiv designer(a)$
Effect:
$Has(a,f,do(composeform(a,f,list(\bar{i}),s))$
7. $subformlist(a,f_s,f,\bar{i})$ – Agent a composes a subform of f consisting of items $\bar{i}$.
Precondition:
$Poss(subformlist(a,,f_s,f,list(\bar{i}),s) \equiv (designer(a) \bigwedge_{j=1}^{n} contains(f,i_j,s)$
Effect:
$Has(a,f_s,do(subflist(a,f_s,f,list(\bar{i}),s))$
$\wedge subform(f_s,f,do(subflist(a,f_s,f,list(\bar{i}),s))$
$\bigwedge_{j=1}^{n} contains(f_s,i_j,do(subflist(a,f_s,f,list(\bar{i}),s))$
8. $subformsubtract(a,f_s,f,\bar{i})$ – Agent a composes a subform of f by removing items $\bar{i}$.
Precondition:
$Poss(subformlsubtract(a,,f_s,f,list(\bar{i}),s) \equiv (designer(a) \bigwedge_{1}^{n} contains(f,i_i,s)$
Effect:
$Has(a,f_s,do(subformsubtract(a,f_s,f,list(\bar{i}),s)) \wedge subform(f_s,f,s)$

This vocabulary is sufficient for specificying a number of attacks on form signing procedures.

3.4 Finding Attacks on Forms

We consider a simple toy example. Suppose that we have produced a form that is intended to collect data about the electronics someone has purchased. There is an item representing a title “Items Purchased.” There are then five items such as “Computer” or “Television” with check boxes. The intention is that the form should be filled out, signed, then returned to the relevant authority.

Here is a high level description of an attack sequence on this form.

1. Form designer makes a form and sends it to A.
2. An intruder intercepts the form.
3. The intruder changes the signing area to not include the title, then forwards the form to A.
4. A checks the appropriate fields and sends the form to the authority.
5. The intruder intercepts the form again, and changes the title to “Items Stolen” and forwards the signed form.

This sequence of actions can be formalized using SitCalc actions from our theory as follows.

1. Use *composeform* and *send*.
2. Intercept using message passing actions.
3. Use *subformsubtract* to remove title. Use *composeform* to re-add outside the signing area.
4. A fills out the form using *check* action.
5. Intercept again, use *subformsubtract* and *composeform* again to produce final form.

In practice, a form designer might not even think it is necessary to put the form title under the signature of the signer. But failing to do so opens up the possibility of this kind of attack, where the entire meaning of a form is change by a malicious intruder. Worse yet, the meaning is changed *after* an honest agent has signed the form. While this particular attack might not be a problem, it does suggest that some form designs might be vulnerable to such attacks. Unfortunately, this kind of attack is very hard to find by hand; it would be much better if it could be found automatically through a translation to GoLog.

4 DISCUSSION

4.1 Future Work

There are two natural directions for future work. First, as mentioned in the previous section, it would be valuable to turn our theory into an executable program for finding attacks. The Golog implementation from (Hunter et al., 2013) can immediately be applied to find attacks due to form interception or mirroring. In order to develop a more complete approach to form validation, we would also like to have the intruder use form modification actions. The challenge is to avoid any infinite loops in which a sub-form is created, then extended, then contracted forever. But this is a standard problem already addressed in the SitCalc tool for protocol verification, so it would be easy enough to use the same methods for the form validation tool. The second obvious direction for future research would be to allow more complex forms with more structure and a wider range of options.

In the long run, it would be ideal to embed our form validation tool in something like the Lotus Forms designer tool. At present, Lotus Forms allows a designer to create a form, but it does not look for any particular attacks. It would be a significant improvement if the designer would look for possible attacks on a form, based on the number of signature fields and the network over which it must be shared.

4.2 Conclusion

We have presented a SitCalc formalization of form signing procedures. We have demonstrated that form signing procedures can be formalized by extending existing SitCalc tools for representing and reasoning about cryptographic protocols. We have noted that this is a problem of important practical significance, which has not been extensively explored using formal methods. It has however been demonstrated elsewhere that key concepts such as commitment and deception can be formalized in a logical framework in which forms are represented as trees, and agents have explicit beliefs. By using epistemic extensions of the SitCalc, the framework presented here could therefore be used to formalize and reason about these more complex notions as well. Logical methods have a great deal to offer in the context of this problem; this paper is just scratching the surface in terms of technical content and practical applications.

REFERENCES

Bertrand, R., Hearn, J., and Lett, B. (1995). The north american pre- and post-processing equipment market: Capturing the benefits and avoiding the pitfalls. Technical report, Strategic Analysis Report, Gartner Group.

Boyer, J. (2005). Enterprise-level web form applications with xfdl and xforms. In *Proceedings of XML 2005 Conference and Exposition*.

Burrows, M., Abadi, M., and Needham, R. (1990). A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36.

Carlucci Aiello, L. and Massacci, F. (2001). Verifying security protocols as planning in logic programming. *ACM Transactions on Computational Logic*, 2(4):542–580.

Dolev, D. and Yao, A. (1983). On the security of public key protocols. *IEEE Transactions on Information Theory*, 2(29):198–208.

Fagin, R., Halpern, J., Moses, Y., and Vardi, M. (1995). *Reasoning About Knowledge*. MIT Press.

Halpern, J. and Pucella, R. (2003). On the relationship between strand spaces and multi-agent systems. *ACM Transactions on Information and System Security (TISSEC)*, 6(1).

Hernández-Orallo, J. and Pinto, J. (2000). Especificación formal de protocolos criptográficos en cálculo de situaciones. *Novatica*, 143:57–63.

Hunter, A. (2012). Structured documents: Signatures and deception. In *Proceedings of the European Intelligence and Security Informatics Conference (EISIC 2012)*, pages 274–277.

Hunter, A., Delgrande, J., and McBride, R. (2013). Protocol verification in a theory of action. In *Proceedings of the Canadian Conference on AI*, pages 52–63.

Levesque, H., Pirri, F., and Reiter, R. (1998). Foundations for the situation calculus. *Linköping Electronic Articles in Computer and Inf. Science*, 3(18):1–18.

van der Hoek, W. and Wooldridge, M. (2002). Tractable multiagent planning for epistemic goals. In *Proceedings of AAMAS-02*,.