SECURITY ANALYSIS OF FEATURE INTERACTIONS

Rui Gustavo Crespo
INESC-ID, Technical University of Lisbon, Lisbon, Portugal
R.G.Crespo@comp.ist.utl.pt

ABSTRACT

Internet applications, such as Email, VoIP and WWW, have been enhanced with many features. However, the introduction and modification of features may result in undesired behaviours, and this effect is known as feature interaction-FI.

In this paper we propose a new approach for the identification of FIs, based on security threats. After building a security model of events and predicates that may exist, with their restrictions, an inconsistency is injected in the post-condition and propagated backwards to the precondition. If successful, the method generates a new feature that raises a FI.

The approach revealed to be successful in the identification of new features that result on FI occurrence for a set of ten widely used Email features.

KEYWORDS

Feature interaction detection, Security threats, Inconsistency rules, Event and predicate constraints

1. INTRODUCTION

Internet applications are being enhanced with many features. A feature is defined as a unit of functionality existing in a system and is usually perceived as having a self-contained functional role [1]. The combination of features may result in undesired behaviours and this problem is known as feature interaction, or FI for short [2].

The FI problem, first identified in circuit-switched networks, has been studied in many Internet applications, such as Email [3], VoIP [4], WWW [5] and networked home appliances [6].

Example 1: Suppose that Bob instructs the Email server to execute the ForwardMessage feature, forwarding all messages to Carl. Suppose also, that Carl subscribes to the AutoResponder, by activating the UNIX vacation program.

A message that Alice sends to Bob is forwarded to Carl. Thereafter, the Email server of Carl accepts Alice's message and sends a notification message to Bob, not to the message initiator (Alice). This result goes against the initial goal of AutoResponder feature, to notify the initiator that Carl is on vacation.

The Email server of Bob, when it receives the notification message, forwards it back to Carl. The Email server of Carl detects a loop, another undesired behaviour, and discards the notification message.

The increasing number of FIs, and the inconvenience they are causing, has led industry and researchers to meet regularly at the Feature Interactions in Telecommunications and Software Systems workshops, eight of which have been held from 1992 to 2005.

Three basic problems have been studied, avoidance, detection and resolution. Avoidance means to intervene at the protocol or design stages to prevent FIs, before features are executed. Detection aims at the identification of FIs, with suitable methods. In the resolution, actions are exercised over already detected FIs.
2. APPROACHES FOR FI DETECTION

Research on FI detection has been focused on methods to represent features and check if a pair of features is likely to interact. This can be done by using one of the many static FI detection algorithms known in the literature. A review of several existing methods is given in [7].

FI detection methods adopt property identification, behaviour approaches or both. Properties analysed in FI detection include inconsistency, releasability and satisfiability. This reactive approach for FI detection reveals the disadvantage that, when a new feature is added or a subscribed feature is updated, a new process of FI detection must be executed. This process may lead to the identification of new interactions, which require the modification of previous resolution practices.

In this paper we propose a new proactive approach to FI, based on the security analysis of features. Given one feature, one set of possible events that may be generated in the system lifetime and the conditions that reveal a FI, we ask ourselves in what cases another feature, existing or hypothetical, make FIs occur with the given feature. This approach has the advantage to highlight, in advance, the FIs that may occur when subscribing new features and take the necessary precautions in advance in the FI resolution methods [8].

3. FEATURE REPRESENTATIONS

In our work, feature specifications are described by two representation schemes, behavioural and analytical. The first is closer to Human reasoning; the second is more suitable for security analysis.

Central to the feature specification are events, or messages, defined as signals generated by a participant or by a feature, that may carry information. In some cases, an event extends another base event with extra information. In these cases, the extra information is stored as arguments positioned after those of base event.

For example, \texttt{send(I,O,T)} event states that a message, initiated by \texttt{I} and last processed by the originator \texttt{O}, is sent to the intended terminator \texttt{T}. The event \texttt{sendE(I,O,T,PK)} extends \texttt{send} event with argument \texttt{PK} indicating the participant, which the public key is used to cipher data. The event \texttt{sendA(I,O,T)}, used in the \texttt{RemailMessage} feature, extends \texttt{send} message with no extra parameters, and states that initiator \texttt{I} is an anonymous address.

3.1. Behavioural scheme

For the behavioural scheme, we propose to use first-order predicate logics [9] to identify properties observed prior and after processing a feature.

A feature specification is a set of quadruple \((E,{\{P\}},E*,{\{R\}})\) where \(E\) represents the trigger event, \(E^*\) represents the set of possibly empty generated events, \(\{P\}\) represents the precondition, the condition that must be satisfied before the features is executed and \(\{R\}\) represents the post-condition, the condition satisfied after the feature conclusion.

Because features must be deterministic, preconditions must be mutually exclusive when different quadruples of the same feature are triggered by the same event.

Example 2: The \texttt{ForwardMessage} feature is triggered by a message sent to the subscriber, \texttt{send(A,B,\_self)}. As result of \texttt{ForwardMessage} execution, a new event \texttt{send(A,\_self,C)} is generated. This new event means a message initiated by \texttt{A}, originated by the \texttt{ForwardMessage} subscriber with \texttt{C} as the intended terminator. The feature’s subscriber defines the participant \texttt{C}.

In \texttt{ForwardMessage} feature definition, no conditions are applied. Hence precondition and post-condition are equal to \texttt{true} and the \texttt{ForwardMessage} behavioural definition is
(send(A,B,_self), true, send(A,_self,C), true)

The FilterMessage feature screens the message initiator, and the message is read only if the initiator does not belong to a "black list". The behavioural definition is:

\[
(\text{send}(A,B,\_\text{self}), \text{interdict(\_\text{self},A)}, \emptyset, \text{interdict(\_\text{self},A)})
\]

\[
(\text{send}(A,B,\_\text{self}), \neg \text{interdict(\_\text{self},A)}, \emptyset, \text{read(\_\text{self},A)} \land \neg \text{interdict(\_\text{self},A)})
\]

3.2. Constraints over events and predicates

In the behavioural specification of features events play a limited role as a trigger for the feature execution and the semantics lie in the feature's conditions. Yet, feature parameters face constraints on their parameter values. Likewise, predicate parameters also face constraints on their values.

Some predicates, such as interdict(A,C), are designated as predicate status because they cannot be modified in the feature execution. Therefore, status predicates, if held in precondition, then it must also be held in post-condition.

In some cases, the feature cannot generate events when the predicate is satisfied in the post-condition. They are referred as blocking predicates. held(A,C) is an example of a blocking predicate.

Table 1 depicts constraints faced by eight events and six predicates used in the specification of ten Email features [3]. [B] and [S] prefixes indicate, respectively, blocking and status predicates.

<table>
<thead>
<tr>
<th>Type</th>
<th>Identifier</th>
<th>Meaning</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event</td>
<td>anon(I,Cod,T)</td>
<td>Anonymous message</td>
<td>I#T</td>
</tr>
<tr>
<td>Event</td>
<td>retrieve(I)</td>
<td>Client retrieves message</td>
<td>true</td>
</tr>
<tr>
<td>Event</td>
<td>send(I,O,T)</td>
<td>Normal message</td>
<td>O#T</td>
</tr>
<tr>
<td>Event</td>
<td>sendA(I,O,T)</td>
<td>Anonymous message</td>
<td>O#T</td>
</tr>
<tr>
<td>Event</td>
<td>sendB(I,O,T)</td>
<td>AddressBook message</td>
<td>O#T</td>
</tr>
<tr>
<td>Event</td>
<td>sendE(I,O,T)</td>
<td>Ciphered message</td>
<td>O#T \land (read(T,I)\rightarrow PK=I)</td>
</tr>
<tr>
<td>Event</td>
<td>sendN(I,O,T)</td>
<td>New message is sent</td>
<td>O#T</td>
</tr>
<tr>
<td>Event</td>
<td>sendS(I,O,T)</td>
<td>Signed message</td>
<td>O#T \land (read(T,I)\rightarrow PK=I)</td>
</tr>
<tr>
<td>Predicate</td>
<td>[S]alias(A,C)</td>
<td>C is alternate address for A</td>
<td>A#C</td>
</tr>
<tr>
<td>Predicate</td>
<td>[B]held(A,C)</td>
<td>C held for later delivery</td>
<td>A#C</td>
</tr>
<tr>
<td>Predicate</td>
<td>[S]interdict(A,C)</td>
<td>C messages are refused</td>
<td>A#C</td>
</tr>
<tr>
<td>Predicate</td>
<td>[S]maps(A,C)</td>
<td>C maps to A</td>
<td>A#C</td>
</tr>
<tr>
<td>Predicate</td>
<td>read(A,C)</td>
<td>A reads message written by C</td>
<td>A#C</td>
</tr>
<tr>
<td>Predicate</td>
<td>[S]registered(A,C)</td>
<td>C is a registered user</td>
<td>true</td>
</tr>
</tbody>
</table>

For example, sendE(I,O,T) message must satisfy two constraints: the originator and terminator participants must be different and, if the message is deciphered by the terminator, the terminator must hold the public key. registered(A,C) predicate does not suffer any restrictions, hence the restriction formula is true.

3.2. Analytical scheme

The maximum number of actors that participate in communication features is three. Therefore, the universe of discourse is limited and quantification over variables may be replaced by a limited set of variable mappings to participants.
We propose to represent conditions by states containing numerical codes, one for each parameter, and events by transitions between states labelled by codes, one for each parameter. The numerical codes are binary exponential powers $2^i$, $i \geq 0$.

Example 3: Consider that Alice subscribes ForwardMessage and FilterMessage features, specified in example 2. Consider also that the feature's environment contains two other participants, named Bob and Charles.

Table 2 depicts one possible coding for the events and predicates. For predicates, we separate positive and negative literals with a minus sign, with positive literal on the left side.

<table>
<thead>
<tr>
<th>Type</th>
<th>Identifier</th>
<th>$A_{FW}$</th>
<th>$B_{FW}$</th>
<th>$C_{FW}$</th>
<th>$A_{FI}$</th>
<th>$B_{FI}$</th>
<th>alice</th>
<th>bob</th>
<th>charles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event</td>
<td>send₁</td>
<td>$2^0$</td>
<td></td>
<td></td>
<td>$2^1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Event</td>
<td>send₂</td>
<td>$2^1$</td>
<td></td>
<td></td>
<td></td>
<td>$2^3$</td>
<td>$2^4$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Event</td>
<td>send₃</td>
<td>$2^2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$2^6$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Predicate</td>
<td>read₁</td>
<td></td>
<td></td>
<td></td>
<td>$2^7$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Predicate</td>
<td>read₂</td>
<td></td>
<td></td>
<td></td>
<td>$2^8$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Predicate</td>
<td>interdict₁</td>
<td></td>
<td></td>
<td></td>
<td>$2^9$-2¹⁰</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Predicate</td>
<td>interdict₂</td>
<td></td>
<td></td>
<td></td>
<td>$2¹¹$-2¹²</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For the coded representation of event send, with 3 arguments, we identify three unary components send₁, send₂ and send₃. Predicates read and interdict hold arity equal to 2, hence both predicates are replaced by two unary predicates. Variables used in ForwardMessage and FilterMessage feature specification hold, respectively, $FW$ and $FI$ indexes.

Figure 1 depicts the graphical analytical representation of ForwardMessage feature, with exponential powers depicted between square brackets.

![Graphical analytical representation of ForwardMessage](image)

Figure 1. Graphical analytical representation of ForwardMessage

$S_A$ and $S_B$ represent, respectively, feature starting and ending states. Both states contain value 0, the representation of condition true.

Coding of different instances of the same event, or predicate, at the same position, requires variable binding. In case the same event, or predicate, at the same condition-pre or post, contains different participants, the process of identifying the analytical scheme fails. Also, the process of identifying the analytical scheme fails in case predicate and event constraints are not satisfied in variable binding.

4. SECURITY ANALYSIS

The security analysis algorithms are based on a model expressing a set of inconsistent rules and the restrictions that events and predicates must always satisfy.

We devised two algorithms for the generation of hypothetical new features that raise a FI: simultaneous walk and side by side walk. The two methods differ on the difference between the number of generated events in the feature under analysis and the inconsistency rule.
4.1. Inconsistent rules

An inconsistency rule is a condition satisfied by the post-condition, the trigger and the generated events.

Table 3 depicts six inconsistency rules, identified with help of experience on FI detection in [3]. The table is, by no means, complete and serves to demonstrate the validity of our approach. The first two rules focus on ciphered messages, and rules I3 and I4 focus on anonymous messages.

<table>
<thead>
<tr>
<th>Reference</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1</td>
<td>send[G](I, self, T) &amp; sendE[G](I, self, T, PK)</td>
</tr>
<tr>
<td>I2</td>
<td>send[G](I, self, T) &amp; sendE[T](I, O, self, PK) &amp; read( self, I)</td>
</tr>
<tr>
<td>I3</td>
<td>sendS[T](I, O, self, PK) &amp; sendA[G](Anon, self, T) &amp; read( self, I)</td>
</tr>
<tr>
<td>I4</td>
<td>send[T](I, I, T &amp; sendA[G](anom, anom, T) &amp; sendS[G](I, I, T, PK) &amp; maps( anom, I)</td>
</tr>
<tr>
<td>I5</td>
<td>send[T](I, O, self) &amp; interdict( self, I) &amp; read( self, I)</td>
</tr>
<tr>
<td>I6</td>
<td>send[T](I, O, self) &amp; send[G](I, self, T) &amp; held(I, self)</td>
</tr>
</tbody>
</table>

T and G suffixes in the identifier differentiate trigger and generated events. For example, inconsistency rule I1 states that a message has been sent both ciphered and in clear.

4.2. Simultaneous walk

Figure 2 depicts the algorithm, where IR refers the inconsistency rule. In the simultaneous walk, the number of events generated in the feature under consideration must be equal to the number of events generated in the inconsistency rule. After the initial coding, the algorithm checks consistency and injects the inconsistency rule events and predicates at the final state.

Restriction rules are checked on event and predicate value binding. On events, it must be possible to incorporate inconsistency rule trigger events into the feature (by equality or extension). Also, if the feature generates events, inconsistency must also generate events (not necessarily equal). On predicates, they cannot exist simultaneously in the feature and negated in the same condition (pre-condition or post-condition) of the inconsistency rule. Also if IR contains more than on predicate in the post-condition, then the feature post-condition must contain at least one predicate.

The new feature pre-condition is built from the status predicates.

Example 4: Consider the ForwardMessage feature, specified in example 2 with analytical codes send\(_1\)(A)=2^0, send\(_2\)(B)=2^1 and send\(_3\)(alice)=2^4 on precondition, and send\(_2\)(alice)=2^2, send\(_3\)(C)=2^3 on post-condition. send(I,O,T) restriction of O\#T must be satisfied, hence B\#alice and C\#alice.

Consider now the events depicted in table 1 and the I2 inconsistency rule.

sendE is a trigger event outside ForwardMessage feature events. The variables are bind I\→A and O→B. The components of sendE are coded as sendE\(_1\)(I)=2^0, sendE\(_2\)(O)=2^1, sendE\(_3\)(alice)=2^4 and sendE\(_4\)(PK)=2^5. read is a predicate outside ForwardMessage feature predicates, and their components are transcribed to read\(_1\)(alice)=2^6 and read\(_2\)(alice)=2^7.

Because ForwardMessage post-condition is true, all inconsistent formulas are viable and we may insert predicate read in the ending state. read is not a status predicate, hence the new feature pre-condition remains equal to true.

Restriction T=PK, due to the presence of predicate read, imposes PK variable in sendE event to be bound to the same participant as T.
The new feature, which results a FI with *ForwardMessage*, is known as *DecryptMessage* and its behavioural definition is

\[ (\text{sendE}(A,B,\_self,\_self), \text{true, } \emptyset, \text{read(\_self,A)}) ) \]

**FindSimultaneousWalk**(*AnalyticalScheme, Events/Predicates, InconsistencyRule*)

**Initialisation:**

For each new IR event and predicate

Code their components.

end for.

Bind IR with the feature and three participants: _self, alice and bob.

**Consistency check:**

If trigger events of feature and IR are different and not extendable, or feature generates events and IR does not generate events, then Exit with failure.

If restriction rules are not satisfied, or a predicate exists in IR and is negated in the feature in the same condition, or the feature contains no predicates in the post-condition and IR contains more than one predicate in the post-condition, then Exit with failure.

**Generate candidate for FI:**

For each new predicate that is sub-formula of the inconsistency rule

Add predicate representation to the post-condition.

If predicate is status, then Add it to the pre-condition.

If predicate is blocking and there are generated events, then Exit with failure.

end for.

**Figure 2. Simultaneous walk**

For the first specification behaviour of *FilterMessage* and inconsistency rule I5, the simultaneous walk generates the basic read service. Features always hold higher precedence than basic services, therefore basic read service is not considered as a security threat.

**4.3. Side by side walk**

Figure 3 depicts the algorithm.

Likewise the simultaneous walk, a check is exercised on the incorporation of inconsistency rule trigger events into the feature trigger events, restriction rules are not satisfied, and a predicate exists in IR and is negated in the feature in the same condition. The checks specific to the algorithm are: feature generated event not being present in the IR generated events and the difference on the number of generated events is not one.
FindSideBySideWalk(AnalyticalScheme,Events/Predicates,InconsistencyRule)

Initialisation:

Consistency check:

If trigger events of feature and IR are different and not extendable, or there is a feature generated event that is not present in the IR generated events, or the difference on the number of generated events is not one, then Exit with failure.

If restriction rules are not satisfied, or a predicate exists in IR and is negated in the feature in the same condition, then Exit with failure.

Generate candidate for FI:

For each predicate that is sub-formula of the inconsistency rule

Add predicate representation to the post-condition.

end for.

For the new generated event, which is not extension of a feature generated event

Install the generated event.

end for.

For each new predicate that is sub-formula of the inconsistency rule

Add predicate representation to the post-condition.

If predicate is status, then Add it to the precondition.

If predicate is blocking, then Exit with failure.

end for.

Figure 3. Side by Side walk

Example 5: Consider the AddressBook feature, with two specification behaviours. In the AddressBook feature, the target address is replaced if it is an alias for the real address, otherwise the message goes through. The behavioural specification is

(send(_self,_self,B), alias(B,C), send(_self,_self,C), alias(B,C))

(send(_self,_self,B), ~alias(B,C), send(_self,_self,B), ~alias(B,C))

Consider the alias(B,C) pre-condition. One possible coding is send1(_self)=2^{0}, send2(_self)=2^{1}, send3(B)=2^{2}, send3(C)=2^{3}, alias1(B)=2^{4} and alias2(C)=2^{4}.

Now consider inconsistency rule I1, depicted in table 3. The generated event send(I,_self,T) is available in the AddressBook feature, with variable bindings of I→_self and T→C (for example, participant charles). We only need to inject, in the interacting feature, the generated event sendE(_self,_self,T,PK).

Because I1 does not contain predicates, the pre-condition and post-condition for new feature are equal to true.

The new feature, which results in a FI with AddressBook, is known as CipherMessage and its behavioural definition is

(send(_self,_self,B), true, sendE(_self,_self,B,_self), true)
4.4. Security analysis tool

A tool, programmed in C with support of Lex and Yacc tools [10], was developed to check the validity of the ideas. The tool, and specifications for ten Email features described in [3], is public available at http://comp.ist.utl.pt/rgc/Generator.zip.

Table 4 depicts the results for the Side by Side walk (SS) and Simultaneous walk (S) methods applied to the specification behaviours. Features are indicated by acronyms, where ICS-“Incoming Call Screening” stands for the FilterMessage feature.

<table>
<thead>
<tr>
<th>Feature</th>
<th>I1</th>
<th>I2</th>
<th>I3</th>
<th>I4</th>
<th>I5</th>
<th>I6</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddressBook1</td>
<td>SS{EM}</td>
<td>S{DM+RM}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AddressBook2</td>
<td>SS{EM}</td>
<td>S{DM+RM}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AutoResponder1</td>
<td>SS{DM+FM}</td>
<td>SS{RM+VS}</td>
<td>S{ICS}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AutoResponder2</td>
<td>S{RM+VS}</td>
<td>S{ICS}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DecryptMessage</td>
<td>SS{DM+FM}</td>
<td>S{ICS}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EncryptMessage</td>
<td>SS{FM}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FilterMessage1</td>
<td>SS{DM+FM}</td>
<td>SS{DM+RM}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FilterMessage2</td>
<td>SS{DM+FM}</td>
<td>SS{RM+VS}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ForwardMessage</td>
<td>SS{EM}</td>
<td>S{DM}</td>
<td>S{DM+RM}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MailHost1</td>
<td>SS{DM+FM}</td>
<td>SS{RM+VS}</td>
<td>SS{FM}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MailHost2</td>
<td>S{DM+RM}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MailHost3</td>
<td>SS{EM'}</td>
<td>S{DM+FM}</td>
<td>S{DM+RM}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MailHost4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MailHost5</td>
<td></td>
<td></td>
<td>S{DM+RM}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RemailMessage1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RemailMessage2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RemailMessage3</td>
<td>SS{EM}</td>
<td>SS{DM+FM}</td>
<td>SS{DM+FM}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RemailMessage4</td>
<td></td>
<td>S{DM+RM}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SignMessage</td>
<td></td>
<td></td>
<td>S{RM}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VerifySignature1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VerifySignature2</td>
<td></td>
<td></td>
<td>S{DM+RM}</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For each of ten features, at least one security threat is identified. There are some specification behaviours, such as RemailMessage, where part of specification behaviours do not face threats under the security model expressed in tables 1 and 3.

Some results are combination of features that do not have practical use. For example, simultaneous walk method identifies that FilterMessage interacts with a combination of RemailMessage and VerifySignature features, under inconsistency rule I3.

\[(sendS(I,O_self,PK), true, sendA(Anon,_self,R), read(_self,I))\]

Other results are modification of existing features. For example, side by side method identifies a hypothetical feature that interacts to the third behaviour of MailHost.

\[(retrieve(A), true, sendE(I,_self,T,PK), true)\]

5. CONCLUSIONS AND FUTURE WORK

In our work, we presented an innovative approach for FI detection. Based on a model of possible events and predicates, with restrictions the parameters must satisfy, and a set of possible inconsistency rules, the security threat approach identifies future FI that a subscribed feature may face.
Our algorithms transcribe feature events and predicates to numerical codes. Other types of codes are possible and algorithms for generation of new features should be encapsulated coding, for example using group theory [11]. Also, research should be conducted to identify memory and computational efficiency of data representation and algorithms.

ACKNOWLEDGEMENTS

This research has been partially funded by the Luso-American Foundation for Development, project 419/2006.

We gratefully acknowledge the members of the Computer Security Laboratory at UC Davis, for the insights that have made this paper possible. In particular, I want to thank Prof. Matthew Bishop for all support. Prof. Luigi Logrippo of Université du Québec en Outaouais has also been a constant source of inspiration.

REFERENCES