Identification of Feature Denial of Services

Rui Gustavo Crespo
Technical University of Lisbon-Dept Electrical Engineering and Computers / INESC-ID
Av. Rovisco Pais 1049-001 Lisboa, Portugal
Email: 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 behaviors, and this effect is known as feature interaction—FI. FI resolution may be implemented by a Feature Manager, which is directed by a set of interdiction formulas. On the other hand, the approach by interdiction may eliminate all features candidate for execution, and this result on a feature denial of service.

In this paper we analyze how feature denial of service can be detected, with a design stage algorithm. The detection algorithm has a time complexity of the quadratic of interdiction formulas.

Index Terms—Feature interaction, resolution by interdiction, identification of feature denial.

I. INTRODUCTION

Mobile systems have been expanding dramatically their computational power, as well transmission capabilities. Thanks to these advances, mobile software platforms now offer many popular Internet applications, such as Email and WWW, as well voice and image services.

Internet applications have been 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]. Forward is an example of a widely-used feature that users subscribe to, in different applications such as Email and WWW, respectively referred as ForwardMessage and Refresh.

A. Feature interaction

The combination of features may result in undesired behaviors 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], networked home appliances [4], VoIP [5] and WWW [6].

Likewise, mobile and multimedia communication services also suffer from the FI problem [7].

Example 1.1: 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.

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 absent.

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 behavior, and discards the notification message.

Mobility, itself, may increase the number of FIs. For example, AnonymousMail feature replaces user address with a pseudonym. If an user moves from one area to another, where the service provider does not mask user addresses, the anonymous status may be lost.

Three basic problems have been studied in FI, avoidance, detection and resolution [8].

Avoidance means to intervene at the protocol or design stages to prevent FIs, before features are executed. Due to Internet distributed characteristic with multiple parties and independent developers, avoidance is not considered here.

Detection aims at the identification of FIs, with suitable methods. In the resolution, actions are exercised over already detected FIs. A review of several existing methods of detection and resolution is given in [9].

B. Feature representation and detection

The choices for representing features are vast, many of them adapted to FI detection approaches. Presentation schemes may be divided into informal and formal methods. The former has been based on scenario based presentation schemes, such as UCM [10]. The latter may be further divided between (i) state machine, such as extended FSM [11] and STR [12], (ii) concurrent schemes, such as programming languages [13], Petri-nets [14], process algebra [15] and event-based [16], and (iii) temporal logics [17].

FI detection methods adopt property identification, behavior approaches or both. Properties analyzed in FI detection include inconsistency, releasability and satisfiability. Four approaches for FI detection have been explored so far. They are (i) simulation oriented, where traces of features running together are scanned [18], (ii) model based, where dedicated tools [13] unfolds system requirements into a transition system and system properties are checked, (iii) theorem proving, where it is verified with tool support [19]) if a conjecture—an unacceptable behavior—is a logical consequence of the hypothesis-statements hold when features are subscribed, and (iv) security-based, where a hypothetical new feature, which interacts, is generated by injecting an inconsistency into the feature postcondition and rolling it back to the precondition [20].
C. Feature resolution

The FI resolver for Internet applications must be distributed, scalable, simple to configure and independent of the feature implementations. One possible solution [21] is based on a dedicated Feature Manager, whose architecture is depicted in figure 1.

A message, incoming or outgoing, may trigger several features \( \text{Feat}_A, \ldots, \text{Feat}_N \). Each feature is translated to one, or more, codes. For example, \ForwardMessage\ feature is coded to \Forward(\_dest)\), \AutoResponder\ is translated to a join of \Accept(\_init)\ and \Write(\_init)\ codes. The application sends to the Feature Manager the codes of feature candidates ready for execution. Feature Manager imposes the satisfaction of a set of Interdiction formulas, in the form

\[
\text{Requests} \land \text{Conditions} \rightarrow \text{Restriction} \tag{1}
\]

The Requests part is a conjunction of feature codes, all of them part of the code translation of features candidate for execution.

Conditions part is a conjunction of predicates or their negations. Predicates reflect user status, such as \busy(\_self)\)-the terminator being busy, and message status, such as \loop(\_self)-message was processed previously by the originator which means it entered in a loop. If necessary, Feature Manager requests information from applications to identify special conditions. For example, FM may collect, from the initiator or from the terminator, permission to process the message.

Restriction is one feature code, to be marked for elimination prior to the selection of feature advise.

Example 1.2: Interaction between \ForwardMessage\ and \FilterMessage\ features is resolved by interdicting the message to be sent forward to the intended destination \_dest\, when subscriber forbids the application to process messages sent by the initiator. Interdiction formula is

\[
\Forward(\_dest) \land \neg \perm(\_init) \rightarrow \IF\Forward(\_dest) \tag{2}
\]

Feature manager must ask to the feature subscriber, \_self\, for the value of \perm(\_init)\ predicate.

Feature coding makes possible to transfer the focus on interdiction formulas from the Requests part to Conditions and Restriction parts. This approach increases interdiction formula readability and, as to be seen in section III-B, eases the complexity of the identification of denial service. For example, there are several versions for the VoIP call forward, such as \FCA-ForwardCallAlways(\_dest)\ and \FCB-ForwardCallBusy(\_dest)\). Both versions are coded by \Forward(\_dest)\, only the Conditions part is different-true for \FCA\ and \busy(\_self)\ for \FCB\.

After Feature Manager eliminates features marked as interdicted, it advises the application to execute one of the survival candidates.

The feature elimination has linear complexity in terms of interdiction formulas. Experiments with James Email server [21] reveal moderate losses of performance and, in some cases, even improve the basic server.

II. DENIAL OF SERVICE

Denial of service-DS has traditionally been seen as a threat coming from the system environment through the network. Usually, denials of service threats take profit of implementation details such as buffer-overflow [22]. The inconvenience to customers, and the financial and life threats that the denial of service is causing, lead many researchers, governmental agencies and industry to direct their attention to the identification of DS causes and how to increase protection for computer-based systems accessed through public networks.

In this paper we focus on another potential cause for denial of service, the elimination approach for FI resolution. In this case, a feature denial of service occurs when one non-empty set of candidates, submitted to the Feature Manager, result in all features being marked for interdiction. Unlike system-based denial of service caused by implementation deficiencies, generated by developers, feature denial of service is caused by deficiencies on FI resolution policies, generated by customers.

To develop an algorithm to identify when feature denial of services occurs, we define the following sets:

- \( F = \{\text{Accept}, \text{Deny}, \text{Forward}, \ldots\} \) is the set of all possible feature codes.
- \( P = \{\text{busy}, \text{perm}, \ldots\} \) is the set of all possible predicates.
- \( I : 2^F \times (2^P \cup \{\text{true}\}) \times F \) is the set of all possible interdiction formulas.

While the number of feature codes is minute, the number of features may be very large. For example, it is estimated telephone systems offer to subscription about eight hundred features [18].

III. DENIAL TO A SET OF FEATURE CANDIDATES

From one arbitrary set of features candidate for execution, we identify \( C_F \subseteq F \) as the subset of codes for the feature candidates.

In this section we describe one algorithm to detect if a set of interdiction formulas reduces the set of feature codes to an empty set. The algorithm is based on the exhaustive search and takes profit on the associativity of set difference.
A. Algorithm

To detect feature denial of service, we start to identify all possible results for the application of interdiction formulas. The results are depicted in a DS tree, with nodes containing \( C_F \) subsets, \( C_F \) at the root, and branches labeled with the interdiction formula and the predicates that must be satisfied. One path, from root to a leave, depicts the results of exercising a sequence of interdiction formulas.

For every interdiction formula such that (First parameter is subset of \( C_F \) and parameters may be unified from root and predicates from root do not contradict),

- Create subnode with \( C_F \setminus \{3^{rd} \text{ parameter}\} \).
- For each formula with same \( 3^{rd} \) parameter, do
  - Create link to subnode with \( 2^{nd} \) parameter.
- If subnode is empty, then
  
  Terminate. /* feature DS found! */
  
  else continue from subnode.

done.

\[ C_F \leftarrow C_F \setminus \{3^{rd} \text{ parameter}\} . \]

/* feature DS inexistent! */

Fig. 2. DS generation algorithm

To make the identification of feature denial of services more efficient, we minimize the number of DS nodes by eliminating duplicate information.

First, the order of element removal from a set is irrelevant. For example, exercising first interdiction formula \( I_1 \) (which results on the removal of \( c_1 \) feature code) then exercising interdiction formula \( I_2 \) (which results on the removal of \( c_2 \) feature code) has the same effect as exercising \( I_2 \) first, then \( I_1 \).

Secondly, we want to depict only once the representation of a subsequence of interdiction formulas. For example, if we have a path of exercising interdiction formulas \( I_1 \) first and \( I_2 \) second, this path already depicts the effects of exercising only \( I_1 \) interdiction formula. Therefore, after creating a path containing only interdiction formula \( I_1 \), only the path containing interdiction formulas \( I_2 \) first and \( I_1 \) second would add more information. The algorithm to generate the DS tree in a depth-first manner [23] is depicted in figure 2.

Figure 3 depicts an example of a DS tree for four feature codes \( F = \{A, \ldots, D\} \). To make the representation more readable, dashed circles represent an eliminated feature code. The DS tree is a transcription of the Pascal’s triangle [24], whose vertices represent combinations of \( F \) elements marking for elimination \( k \) at a time, \( C_F^k \).

The identification of a feature DS requires the expansion of all non-empty DS leaves, by exercising figure 2 algorithm. The expansion holds no predicates, or holds predicates established in the path from an upper level node to the DS leaf.

Fig. 3. DS tree representation for \( F = \{A, B, C, D\} \) combinations

Example 3.1: Consider ForwardMessage and FilterMessage as candidates for feature execution. Consider also interdiction formula

\[
\text{Deny}_\text{init}(\text{init}) \land \text{emergency}_\text{init}(\text{init}) \rightarrow \text{IDeny}_\text{init}(\text{init}) (3)
\]

which forbids the denial of emergency calls, plus (2) interdiction formula.

Figure 4 depicts the DS tree on the left, with \( D \) standing for \( \text{Deny}_\text{init}(\text{init}) \) and \( F \) standing for \( \text{Forward}_\text{dest}(\text{init}) \) feature codes. Branches are labeled with the exercised interdiction formula plus the predicate that must be satisfied, or negated.

The root contains the two feature codes, \( \text{Forward}_\text{init}(\text{init}) \) and \( \text{Deny}_\text{init}(\text{init}) \). We first select (2) interdiction formula and create a branch with sub-node containing \( \text{Deny}_\text{init}(\text{init}) \) and labeled with \( \neg \text{permission}_\text{init}(\text{init}) \) predicate. DS tree expansion exercises (3) interdiction formula and creates a branch to an empty leaf. We conclude then a feature denial occurs if
To avoid the denial of service, emergency should be considered as a message and a feature code. The interdiction formula should then be replaced by a new formula (4)

\[ \text{Emergency}(\text{init}) \rightarrow I \text{All} \backslash \text{Emergency}(\text{init}) \quad (4) \]

\[ \text{Deny}(\text{init}) \land \text{Emergency}(\text{init}) \rightarrow I \text{Deny}(\text{init}) \quad (5) \]

(4) formula is expanded on both sides of implication all feature codes different to Emergency and becomes (5) formula. The resulting DS tree is depicted on the right of figure 4. □

B. Efficiency in the Identification of Feature Denial

First we identify the cost of generating the DS tree, when each feature code is interdicted by one formula only. In the worst-case there is no feature denial of service and all DS tree is generated. The sum of a row \( N \) in a Pascal triangle, \( \sum_{k=0}^{N} C_k^N \), is equal to \( 2^N \) [24]. The sum of a geometric series is also an exponential power, hence the worst-case complexity of generating DS tree is \( O(2^F) \).

We now turn the attention to the cost of identifying predicate contradictions. To increase the efficiency of calculating a contradiction on a predicate conjunction, we store the representation of predicates as pairs of prime numbers plus a sign indicating if predicate is negated. The check for contradiction on the conjunction of a new predicate is simply done by verifying if the new predicate already exists in negative form.

On average, for each feature code there are \( P/F \) different interdiction formulas. The number of branches from Pascal triangle rows \( N \) to \( (N+1) \) is equal to \( N+1 \). The sum of a linear series is quadratic [24], hence the worst-case complexity of generating DS tree is \( O(2^F F^2) \). Because the number of feature codes is very small, we may consider the number of predicates is a fraction of the number of interdiction formulas. Hence, the worst-case cost of identifying feature denial of service is \( O(2^F F^2) \).

IV. CONCLUSIONS AND FUTURE WORK

In this paper we analyzed how and interdiction based approach for feature resolution may result in feature denial of services. The feature denial of services is not permanent and occurs only in some particular combinations of features candidate for execution. Yet, the possible case must be known in advance so managers may modify the configuration of FI resolver that suits customers and guarantees the application availability.

The cost of identifying feature denial of service is quadratic in the number of interdiction formulas.

To decrease search costs, other strategies for identifying feature denial of service, such as incremental updates of interdiction formulas, should also be analyzed.

ACKNOWLEDGMENTS

This research has been partially funded by the Portuguese Science and Technology Foundation, project PTDC/EEA-ELC/66472/2006.

We also gratefully acknowledge Prof. Luigi Logrippo of Université du Québec en Outaouais, Canada as a constant source of inspiration.

REFERENCES


