Improving Process Robustness by Identifying Vulnerabilities using Fault Tree Analysis

Huong Phan, George Avrunin, Lori Clarke, and Leon Osterweil

University of Massachusetts Amherst, MA 01002, USA
{hphan,avrunin,clarke,ljo}@cs.umass.edu
http://laser.cs.umass.edu

Abstract. This paper presents a systematic, incremental approach to identifying vulnerabilities in models of complex processes. The approach builds upon Fault Tree Analysis (FTA), but enhances it to deliver superior results. FTA has been applied to process models to determine what combinations of events can lead to a specified *hazard*—an undesired process outcome. Such combinations, called *cut sets*, provide valuable insights into process vulnerabilities, and thus, are useful for improving process robustness. Manual FTA can be tedious and error-prone. Results of previous automated FTA work, however, are incomplete as they fail to recognize some vulnerabilities. Also, they often do not provide enough information to understand the real vulnerabilities, or they provide too many cut sets to be helpful, especially when processes are large and complex. Our approach takes into account processes data and control dependences to generate more thorough results. It supports selective vulnerability exploration by initially presenting users with cut sets for a high-level, and thus smaller, fault tree; users can then select a cut set for more detailed analysis, culminating in *concrete scenarios* which show how events in the cut set could lead to the hazard. Our approach also produces more precise results by automatically eliminating inconsistent and spurious cut sets.

Key words: fault tree analysis, vulnerability identification, process robustness improvement

1 Introduction

Using Fault Tree Analysis (FTA) to identify vulnerabilities in processes is one way to help improve process robustness. FTA has been widely used to study system safety in various domains such as aviation, nuclear power, chemical engineering, etc. [1]. Chen adapted this technique to analyze process models [2, 3]. As manual fault tree construction and analysis are tedious and error prone, he developed systems that automatically derive a fault tree from a process model given a *hazard*—an undesired situation; and then automatically compute *cut sets*—combinations of events that lead to the hazard. Such cut sets are useful in understanding vulnerabilities, and thus, can suggest process modifications

that lead to process improvements. Once modifications are made to the process model, the analysis can be performed again with minimal effort to validate that such modifications are effective.

Chen's FTA approach has been shown to be useful in some preliminary work to analyze process models in the medical and election domains [3, 4, 5]. It does, however, have some drawbacks. This approach uses data flows and a limited set of control flow primitives in a process model to trace back to possible causes of a hazard, but does not consider control dependences, and therefore fails to identify some vulnerabilities. Moreover, cut sets that are computed often do not provide analysts with enough information to readily gain sufficiently deep understandings of the nature of underlying process vulnerabilities. In addition, when processes are complex with large numbers of steps and parameters, this approach typically produces huge fault trees and numerous cut sets that are often difficult to understand.

In our work we present a systematic, incremental approach to using FTA to identify process vulnerabilities. We still use templates to automatically derive fault trees from process models and we compute cut sets as in Chen's approach, but we exploit more process details to deliver sharper analytic results, and use an iterative approach to help users gain deeper understandings more easily. The contributions of this work are as follows. First, we produce more comprehensive results, and therefore identify cut sets that were not previously identified, by considering all control dependences together with all data dependences. Second, we reduce users' efforts in interpreting analytic results by supporting selective incremental exploration of the fault tree. We initially present users with cut sets derived from a high-level, and thus relatively smaller, fault tree. Users can then select a cut set as the basis for more detailed analysis. This analysis generates an elaborated fault tree that focuses on only the cut set of interest, and produces as its final result concrete *scenarios* — a scenario is a process execution path that contains all the events in a cut set, showing how the events in the cut set could lead to the hazard. Third, we improve the precision of the results by automatically removing inconsistent and spurious cut sets.

The rest of the paper is organized as follows. We first briefly describe FTA in section 2. In section 3 we present our approach with the example of a small process "issue ballot". In section 4 we discuss our preliminary results of applying the approach to two larger processes, "count votes" and "blood transfusion", and end with a conclusion.

2 Background: Fault Tree Analysis

FTA is a deductive, top-down analytic technique used in various domains to study hazards. FTA starts with the analyst identifying a hazard, which is an undesired system state. FTA works backward from this hazard to produce a *fault tree*—a graphical representation of the various combinations of events that lead to the hazard.

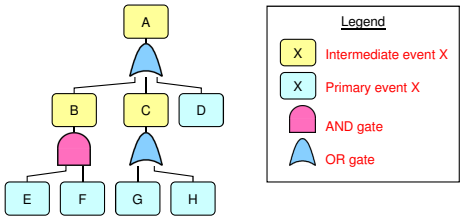

Fig. 1. Fault Tree Example: A occurs if B, C or D occurs; B occurs if both E and F occur. Four cut sets: {E,F}, {G}, {H}, and {D}.

Events and *gates* are the basic elements of a fault tree (Fig. 1). The event at the root (top) of the tree represents the hazard. There are two types of events, *intermediate events*, which are elaborated, and *primary events*, which are not further elaborated. A gate connects one or more input events (below) to a single output event (above). We consider two types of gates: (1) AND gates specifying that the output event occurs only if all input events occur (inputs are assumed to be independent), and (2) OR gates specifying that the output event occurs if any input event occurs. Given the fault tree, simple Boolean Algebra can be used to compute *cut sets*—sets of primary events that when all occur will cause the hazard. Cut sets indicate potential vulnerabilities, which are flaws or weaknesses in a system's design, implementation, operation, or management that could potentially allow the hazard to occur.

3 Our Approach: Using FTA to Identify Vulnerabilities

This work is built upon Chen's template-based FTA approach that derives fault trees from rigorously defined process models. Even though Chen's FTA implementation generates fault trees only from processes that are defined in the Little-JIL process definition language [6], his approach could be adapted to generate fault trees from processes defined in other languages that incorporate sufficient data flow and control flow semantics. Our approach is similarly independent of the choice of process language, although this paper describes its application to a very simple control flow graph (CFG) representation of a process.

In this work, a hazard is defined to be an incorrect artifact input to or output from a specific step or activity of the process. Predefined templates exploit the process's control and data flows to trace back through the process representation to identify where incorrect step or activity performance could lead to the hazard. The templates are applied iteratively to elaborate all intermediate events until all intermediate events have been fully elaborated, resulting in a fault tree whose leaves are all primary events. Our new approach defines more templates, categorized into *simple* and *detailed* templates. The simple templates are used to derive a fault tree at the high level. It is only when users select a specific cut set from the initial high-level fault tree that the detailed templates are used to get an elaborated fault tree, as described later in this section.

Fig. 2 shows two of the simple templates we use in our new approach. The first one is used to elaborate the intermediate event of type *"o output from S*

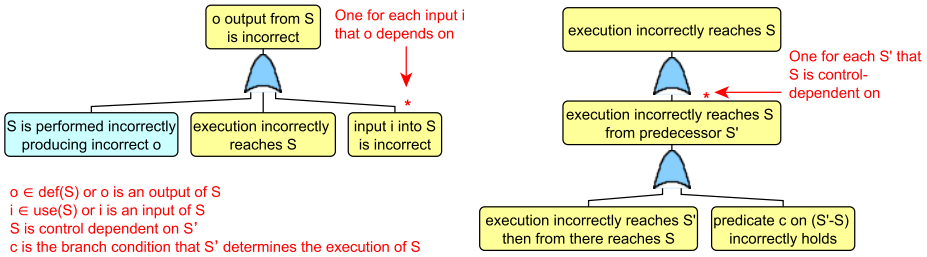

Fig. 2. Simple templates for event of types *"o output from S is incorrect"* (left) and *"execution incorrectly reaches S."* (right)

is incorrect" – S is a CFG node and o is one of its outputs. This event can be caused by either S being performed incorrectly; execution incorrectly reaching S; or one input of S being incorrect [1]. The last two events are intermediate events and therefore can be further elaborated using other templates.

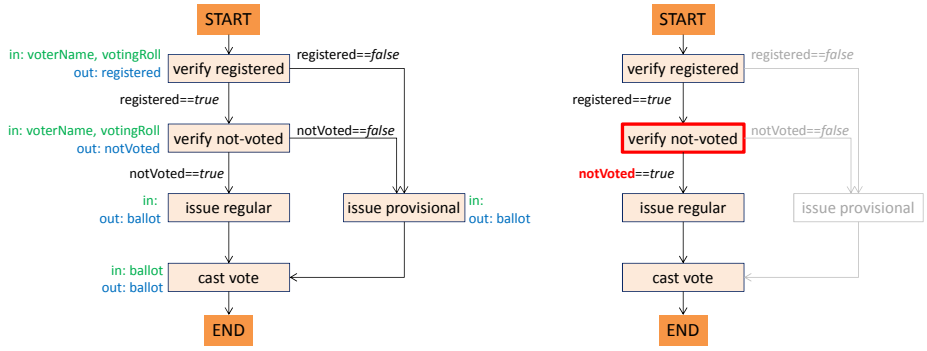

Fig. 3. Left: CFG of the "issue ballot" process. Right: example scenario of how a registered voter, who already voted, gets a regular ballot – *"verify not-voted"* is highlighted RED because the step is performed incorrectly producing incorrect *notVoted*.

Fig. 3 (left) shows a CFG representation of "issue ballot", a very much simplified portion of an election process. As an example of our approach, consider the hazard *"ballot input into 'cast vote' is incorrect"*, we can initially automatically derive a high-level fault tree from this process and compute its cut sets. Besides obvious single-event cut sets such as { *" 'issue regular' is performed incorrectly producing incorrect ballot"*}, there are more complicated cut sets,

[1] As mentioned in the introduction, Chen's FTA does not consider control dependences – its templates do not consider the case *"execution incorrectly reaching S"*.

e.g., {*"verify not-voted" is performed incorrectly producing incorrect notVoted, notVoted==true*} [2].

Our new approach allows users to *zoom in* on a specific cut set by

1. creating a projection of the initial fault tree that keeps only events relevant to this cut set;
2. applying detailed templates to this projected fault tree to derive a more elaborated fault tree, called *focused-elaborated fault tree*; and then
3. automatically computing the new cut sets of the focused-elaborated fault tree and generating concrete scenarios showing ways that the events in the cut sets can occur and how they can then lead to the hazard.

With the above mentioned cut set, the projected fault tree reveals that because *notVoted* is incorrectly true, the process execution incorrectly reaches the step *"issue regular"*, therefore the output *ballot* from that step is considered incorrect. But the analyst might then want to understand better how the execution can ever reach *"verify not-voted"* in the first place. We can facilitate the understanding by applying our new detailed templates to derive the focused-elaborated fault tree, which in turn has new elaborated cut sets, e.g. {*registered is correctly true, "verify not-voted" is performed incorrectly producing incorrect notVoted, notVoted==true*}. This final cut set gives the analyst a better idea of how the hazard arises. In this case, it could be a collusion between a malicious registered voter (hence *registered correctly true*), who already voted, and the agent performing *"verify not-voted"*, who deliberately "verifies" that this voter has not voted, so that the voter can get a regular ballot again (and again) (Fig. 3, right).

4 Preliminary Results and Conclusion

We have applied our new approach on processes of various sizes: the small size "issue ballot" process as descibed above, a medium size process "count votes" (CFG: 48 nodes, 48 edges) with the hazard of incorrect total vote counts being reported, and a larger process "blood transfusion" (CFG: 315 nodes, 338 edges) with the hazard of incorrect blood being transfused to a patient. As expected, this new approach provides more scenarios showing how the hazards may arise, and the analysis results are presented incrementally—from more abstract to more detailed—so that the analysts are not overwhelmed with too much information.

We continue to develop the FTA tool that implements our approach. We also continue to evaluate the approach by applying it on different process models, letting domain experts use the tool and collecting their feedbacks.

We believe that this work provides a superior way to identify process vulnerabilities and therefore helps to improve process robustness.

[2] Chen's FTA produces only two cut sets: {*" 'issue regular' is performed incorrectly producing incorrect ballot"*} and {*" 'issue provisional' is performed incorrectly producing incorrect ballot"*}.

Acknowledgement

This research was partially supported by the U.S. National Science Foundation (NSF) under Award Nos. IIS-1239334 and CNS-1258588 and the U.S. National Institutes of Science and Technology (NIST) under grant 60NANB13D165.

References

1. C. A. Ericson II, "Fault Tree analysis - A History," in *17th International System Safety Conference*, 1999.
2. B. Chen, G. S. Avrunin, L. A. Clarke, and L. J. Osterweil, "Automatic Fault Tree Derivation from Little-JIL Process Definitions," in *Software Process Change*, ser. Lecture Notes in Computer Science, Q. Wang, D. Pfahl, D. M. Raffo, and P. Wernick, Eds. Springer Berlin Heidelberg, Jan. 2006, no. 3966, pp. 150–158. [Online]. Available: http://link.springer.com/chapter/10.1007/11754305_17
3. B. Chen, "Improving Processes Using Static Analysis Techniques," Ph.D. dissertation, University of Massachusetts, Amherst, MA 01003, USA, Sep. 2010.
4. H. Phan, G. S. Avrunin, L. A. Clarke, O. J. Leon, and M. Bishop, "A Systematic Process-Model-based Approach for Synthesizing Attacks and Evaluating Them," in *Presented as part of the 2012 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections*. Berkeley, CA: USENIX, 2012. [Online]. Available: https://www.usenix.org/conference/evtwote12/workshop-program/presentation/Phan
5. B. I. Simidchieva, S. J. Engle, M. Clifford, A. C. Jones, S. Peisert, M. Bishop, L. A. Clarke, and L. J. Osterweil, "Modeling and Analyzing Faults to Improve Election Process Robustness," in *Proceedings of the 2010 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE '10)*, 2010.
6. A. Wise, "Little-JIL 1.5 Language Report," Uinversity of Massachussetts Amherst, Tech. Rep., 2006.