Thesis on Informatics and System Engineering c54

Efficient Semantics of Parallel and Serial Models of Attack Trees

Aivo Jürgenson

TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology
Department of Informatics

Dissertation was accepted for the defense of the degree of Doctor of Philosophy in informatics and system engineering on 12th May, 2010.

Supervisors:

Prof. Dr. Ahto Buldas, Chair of Information Security, Department of Informatics, Faculty of Information Technology, Tallinn University of Technology

Dr. Jan Willemson, senior researcher, Cybernetica AS

Opponents:

Prof. Dr. Sjouke Mauw, Chair of Security and Trust of Software Systems, Faculty of Science, Technology and Communication, University of Luxembourg

Prof. Dr. Tanel Tammet, Chair of Network Software, Department of Computer Science, Faculty of Information Technology, Tallinn University of Technology

Defense of the thesis: 21st June, 2010

Declaration: Hereby I declare that this doctoral thesis, my original investigation and achievement, submitted for the doctoral degree at Tallinn University of Technology, has not been submitted for any academic degree.

/Aivo Jürgenson/

This thesis was typeset with LaTeX using Robert Slimbach's MinionPro typeface.
ISSN 1406-4731
ISBN 978-9985-59-999-0

INFORMAATIKA JA SÜSTEEMITEHNIKA C54

Ründepuude paralleel- ja jadamudelite efektiivsed semantikad

Aivo Jürgenson

Contents

Introduction

When using information systems for critical tasks, for example, controlling and managing industrial processes or handling valuable information, such as e-votes, bank transactions, transmitting state secrets or business sensitive information, information security becomes very important. In general, the owners and users of information systems would like to ensure certain levels of confidentiality, integrity and availability of both information and information systems, so that their own targets could be met. The trouble is that information system components sometimes have vulnerabilities. Threats, both natural and human-made, will sometimes take advantage of those vulnerabilities and will affect the information security targets and users will then fail to achieve their own objectives.

To combat those threats, various security measures are usually deployed. Starting from physical security measures, continuing with organizational and procedural measures and also including IT security measures, such as authentication, access control, encryption, digital signatures, redundancy, sand-boxing, logging, and various other technical measures. They all tend to decrease the risk of materializing threats, however, in the relatively young field of information security, the effectiveness of those security measures is usually not certain. Sometimes mistakes are made in the deployment, sometimes the measures themselves contain bugs and vulnerabilities and the resulting total security situation is not usually known. It still might happen that a skillful attacker might be able to successfully attack us.

It is currently difficult to quantify how hard it is to attack the information system.

Cryptography, a much more rigorous discipline, has concepts of those kinds of guarantees mostly because encryption has been used for ages to protect the military and government secrets during the transmission and huge resources have been applied on one side to read those encrypted messages and on the other side, to make this process harder. Nowadays, we even have mathematical proofs about the *strength* of various cryptographic methods, e.g., the problem of breaking the security of public key encryption can be *reduced* to the problem of factorization of very large numbers. Because it seems that this factorization problem is very complicated, we can therefore say that breaking the public key encryption is a very hard problem as well.

However, most of the cryptographic methods simply transform one kind of security problem into another, e.g., strength of public key encryption depends on the secrecy of private keys and security of SSL/TSL technology depends on users actually paying attention to and understanding browser UI and security alerts. Those assumptions can still be attacked outside the area of cryptography, but within the scope of general information security. Therefore, we cannot just take mathematical proofs from cryptography and rely on *reductionist security* to find out if the information system is secure against practical, real world attackers.

Some other fields use the science of risk assessment to find out which threats are more likely to materialize and how much attention should be spent on managing these. For example, the insurance industry is mostly successful in this business. Usually, they can measure the risk by multiplying the probability of an event with the amount of losses associated with a single event and get an expected loss.

However, the insurance industry usually deals with natural threats and they have a very large body of statistical information on their side. They can simply look up from a database how *frequently* the floods have occurred in that part of the country where the client has built his or her house and then make an informed decision about the premium of the insurance contract. In information security, such kind of data is not (yet) available because the field itself is still very quickly evolving and secondly, the victims of security incidents tend to usually hide this fact and embarrassing details from the general public.

Therefore, it is not possible to simply use frequentist approach for quantifying the probability of complex security incidents.

Engineers also have the same problem, because in their field the reliability of the products is the main concern. Even though they could build a sample of test products and then stress test them and then find out how likely they are to fail in the operational conditions, this becomes rather expensive with larger machines. You cannot just build hundreds of space shuttles and then see if there is a fault in the design or in the selection of components. To find out how likely larger machines are to fail, or how reliable they are before actually building them, engineers have created a methodology called *fault tree analysis*. Multiple ways of how the machine or components could fail are put to the *fault tree* and then connected together with a different kind of logic gates. The resulting tree describes the dependancies between fault events and with the failure probabilities of the individual components, the probability of the total system failure can be simply computed.

The same approach is applied to information security where complicated, multi-stage attacks against information systems are usually composed from simpler attacks. The combined attack could be described by an attack tree that is essentially the same thing as a fault tree. If corresponding security metrics could be assigned to the attack tree leaves and there is a valid computation model, the attacker's success probability, or perhaps even attacker's expected utility could be computed from the attack tree. This would allow the security people and software designers and information system owners to speak with each other about quantified security and make informed decisions concerning the kind of security measures to be applied and built into systems.

The attack tree analysis has been studied for almost 20 years and applied to the real life security analysis as well, however, most of the publications are not based on solid theoretical foundations and economic models, which would allow the attack tree analysis to be used in a company along with other economic considerations.

In this thesis, it is shown that some of the current attack tree models are not usable and do not produce consistent results. Instead of them, two new models are proposed, which allow the following behavior of attackers. In the first model, the attacker can choose the list of elementary attacks to be performed and the attacker's expected utility is then computed based on the assumption that all elementary attacks are performed in parallel. In the second, more realistic model, the attacks are performed sequentially and the attacker can make decisions in between if it is worth trying the next attack, or not. Computing the attacker's utility uses multiple parameters of attack tree leaves, such as the success probability of the attack, cost of the attack and excepted penalties, which are also used in the model of economically motivated crimes.

Compared to previous models, this is a significant advancement of attack tree analysis. The present thesis is based on the following three academic papers written in the period from 2008 until 2010.

I A. Jürgenson and J. Willemson. Computing exact outcomes of multi-parameter attack trees. In R. Meersman and Z. Tari, editors, *On the Move to Meaningful Internet Systems (OTM 2008)*, volume 5332 of LNCS, pages 1036–1051. Springer-Verlag, 2008.

The author contributed to this paper with the implementation of the computation model in the programming language, running the performance tests and collaborating on the parallel attack tree model development.

II A. Jürgenson and J. Willemson. Serial model for attack tree computations. In D. Lee and S. Hong, editors, *Revised Papers from 12th International Conference on Information Security and Cryptology (ICISC 2009)*, volume 5984 of LNCS, pages 118–128. Springer-Verlag, 2010.

The author collaborated on the serial attack tree model development.

III A. Jürgenson and J. Willemson. On fast and approximate attack tree computations. In J. Kwak, R. H. Deng, Y. Won, and G. Wang, editors, *Information Security Practice and Experience, 6th International Conference (ISPEC 2010)*, volume 6047 of LNCS, pages 56–66. Springer-Verlag, 2010.

The author's contribution to this paper was the implementation of the parallel model optimizations, contributions to the genetic algorithm model, implementing and running the performance and accuracy tests.

In Appendix C of the thesis, copies of papers I–III are included on pages 109–155.

The outline of the thesis is the following. Chapter 1 on page 17 and Chapter 2 on page 27 give the overview about the current state of the art in the field of security modeling and quantitative risk assessment with attack trees. Existing research results allow us to perform security analysis and estimate roughly if the attack is feasible or not, but they have several shortcomings. We list those problems and give corresponding counter-examples in Section 2.8 on page 39.

The main result of this thesis is the removal of those shortcomings and presenting the real-life usable methodology to analyze the feasibility of the attack from an attacker's viewpoint. The thesis studies two attack tree models, which allow us to model attackers of different behavior. The first model is presented in Chapter 3 on page 43 and it requires an attacker to decide on the list of attacks before the start of the campaign and then all attacks are tried in parallel. The second model is presented in Chapter 4 on page 65 and that allows an attacker to start attacking and then adaptively decide during the campaign which attacks will be tried next based on the success or failures of preceding attacks.

These models allow a security analyst to find out the expected utility of the attacker. If the utility is positive, we can conclude that there is an effective attack against the current system. If the utility is negative or zero, the system may be secure against rational attackers. However, the exact computing algorithms all have exponential complexity and they can handle in reasonable time only rather small attack trees. To allow processing much larger attack trees constructed in the real-life security analyses, approximate algorithms are presented for both models in Section 3.7 and 4.5.

Acknowledgements

The author has been partly financially supported by the Estonian Science Foundation grants 7081 and SF0012708s06, the Estonian Doctoral School in ICT, Elion Enterprises Ltd., and, last but not least, Cybernetica AS, one of the few private research companies in Estonia.

I am grateful to my supervisors, Ahto Buldas and Jan Willemson. Without Ahto I would not have started studies of the attack trees and without Jan I would not have finished my thesis. I would like to also thank Margus Niitsoo and Alexander Andrusenko, who worked in the same field and shared many ideas and comments.

Abbreviations and Symbols

Abbreviation	Definition
BGP	Border Gateway routing Protocol
CNF	Conjunctive Normal Form of the Boolean formulas
DNF	Disjunctive Normal Form of the Boolean formulas
DPLL	Davis-Puntam-Logemann-Loveland algorithm to find out if the Boolean formula is satisfiable
RDAG	Rooted directed acyclic graph
RFID	Radio Frequency Identification systems
SAT	Satisfiability problem of Boolean formulas
#SAT	Problem of counting the number of solutions to the SAT problem
SCADA	Industrial process control systems
SQL	Structured Query Language
UML	Unified Modeling Language
WTLS	Wireless Transport Layer Security
WAP	Wireless Application Protocol

Symbol	Definition
$\mathcal{T}$	Attack tree $\mathcal{T}$ with AND-nodes, OR-nodes and set of leaves $X = \{X_1, \ldots, X_n\}$ and parameter Gains
$\mathcal{F}$	Boolean formula $\mathcal{F}$ defined by the syntax tree of the attack tree $\mathcal{T}$
X_i	Leaf (elementary attack) X_i of the attack tree $\mathcal{T}$ with parameters p_i and $\mathsf{Expenses}_i$

continued on the next page...

Symbol	Definition
$\mathsf{Expenses}_i$	Expected cost of launching the elementary attack X_i, which includes the preparation costs and expected penalties
p_i	Probability of succeeding with the elementary attack X_i
Gains	Reward of an attacker, when the attack tree $\mathcal{T}$ is realized by the attack suite S
S	Attack suite $S \subseteq X$ of the elementary attacks in the attack tree $\mathcal{T}$
(S, σ)	Ordered attack suite, where $S \subseteq X$ and permutation σ defines the order of elementary attacks $\langle X_{\sigma(1)}, \ldots, X_{\sigma(m)} \rangle$
σ	Permutation σ is the bijective function $\alpha : \{1, \ldots, m\} \rightarrow \{1, \ldots, m\}$
$\mathsf{Outcome}_S$	Outcome value of the attack suite S
$\mathsf{Outcome}_{S,\sigma}$	Outcome value of the ordered attack suite (S, σ)
p_S	Probability of $\mathcal{F} = \mathsf{true}$ after executing the attack suite S
p_σ	Probability of $\mathcal{F} = \mathsf{true}$ after executing the ordered attack suite (S, σ)
$p_{\sigma,i}$	Probability that during the execution of ordered attacks $\langle X_{\sigma(1)}, \ldots, X_{\sigma(m)} \rangle$ the elementary attack $X_{\sigma(i)}$ will be tried
$\mathfrak{S}_m$	The symmetric group with a degree m

Chapter 1

Security Modeling

1.1 Building Models of Security

In science, when studying some particular problem, it is a common to build idealized and somewhat simpler models and then try to understand the phenomenon by applying and testing those models on real life situations. Frigg and Hartmann write in [30]:

Scientists spend a great deal of time building, testing, comparing and revising models, and much journal space is dedicated to introducing, applying and interpreting these valuable tools.

It would be natural that security researchers are also following the same lead and are trying to model the organizations, information systems, attacks and faults from the security aspect.

One of the earliest papers on general security modeling is [70] by Schumacher and Roedig from 2001 and a book [69] by Schumacher from 2003. They build on the success of the software development patterns in the software engineering field and argue that the same kind of approach must be followed in security engineering as well. They identify several possible options for representing those security patterns, such as human readable security policies, Common Criteria standards and also attack trees.

More recently, Miede *et al.* are proposing a generic metamodel in [54] for IT security, capturing both its major concepts and relations of those concepts to each other. Their model includes concepts, such as threat, attack, attacker, attack tool and they also reach a conclusion that quantitative and qualitative attack metrics are useful concepts. They are supported by Trivedi *et al.* who are linking together the security models and dependability modeling concepts in [75]. Nicol *et al.* also compare security models and dependability models in [56], but they come into conclusion that security models should include additional properties which cannot be directly mapped to dependability concepts.

Baadshaug *et al.* study the current approaches to graphical security modeling in [5]. They choose the Vulnerability Inspection Diagram (VID) methodology for their usability and performance testing experiments. The working hypothesis is that specialized security modeling tools will outperform general modeling tools when creating security models. They modify the existing SeaMonster security modeling tool to support VIDs and assign modeling tasks to ten students who are using SeaMonster and general Microsoft Visio tools to solve them. The experiment results show that special security modeling software indeed provides better results and is therefore useful and required.

Tøndel *et al.* combine two security modeling techniques: misuse case diagrams and attack trees in [76]. A similar kind of a merged technique to specify software security requirements has been proposed in [32] by Gandotra, Singhal, and Bedi. The approach was first suggested by Diallo *et al.* [22] in 2006, when they compared the Common Criteria security specification model, misuse cases and attack trees. Misuse cases are analogues to use-cases from the Unified Modeling Language (UML). Where use-cases describe those useful and required activities, which the legitimate users of the system need to able to perform, the misuse cases list those actions which we would like to prevent from happening, e.g., the malicious attacks. Creating attack trees to give more detailed descriptions of how the complicated misuse cases might be achieved, the security models and software requirements become more easily understandable by developers.

The usability of the misuse cases and attack trees as separate techniques were studied by Opdahl and Sindre in [57]. In two experiments, 28 and 35 participants solved threat identifications tasks, first with misuse cases and then with attack trees. The main finding of the study was that attack trees were more effective for finding threats, in particular when there was no pre-drawn use-case diagram to help with identifying the misuse cases.

The research of how to model the security seems to be well underway and studies and experiments show that the attack tree analysis in particular is quite a promising method to apply.

1.2 Fault Tree Analysis

Already in the 1960s and 1970s, the system safety and reliability studies in the U.S. Air Force, Boeing and the nuclear power industry used the ideas of component dependancies and the fault trees. Various papers and courses were put together to a single handbook [79] that was published in 1981 and updated [80] by NASA in 2002. It turns out that the fundamental concept of attack trees can be traced back to the fault tree analysis.

Fault tree analysis is a simple analytical technique, where the undesired state (from the safety viewpoint) of the system is chosen as a starting point and then the system and the environment is analyzed to find all of the possible ways in which the undesired state can happen. The fault tree is the graphical representation of this analysis. The faults could be natural disasters, component failures, personnel mistakes and also deliberate attacks. During the analysis, only those events and system components are taken into account which lead to the realization of the root of the fault tree, the undesired state. Therefore, it is not the model of the complete system itself, but only that of the currently relevant failures and components.

1.2.1 Fault Tree Elements

Fault tree analysis makes an important distinction between faults and failures. Failure is specific malfunctioning of a component of the system. However, there can also be situations where the component is functioning perfectly well, but the system is still in an unsatisfactory state because of some other component failures or some environment issues. Therefore, the fault tree analysis uses a more general word *fault* to describe any kind of events.

Fault events are connected together with different types of logic gates that show the relationship between the *lower* events and between the *higher*, intermediate events. The *higher* event is the output of the gate, the *lower* events are the inputs to the gate. Different gates can be used to describe this relationship in a more formal way. A sample of the fault tree is presented in Figure 1.1 on page 21.

The handbook [79] defines the following gates.

- AND – output fault occurs if all of the input faults occur. AND-gate implies the causality connection between the input faults and the output fault.
- OR – output fault occurs if at least one of the input faults occurs. The causality between input faults and output faults are never passed through the OR

gate.

- XOR – output fault occurs if exactly one of the input faults occurs.
- PRIORITY-AND – output fault occurs if the input faults occur in a specific sequence.
- INHIBIT – output fault occurs provided that some condition is satisfied. The condition is not modeled as an event but something different.

The handbook writers conclude that the XOR, PRIORITY-AND, INHIBIT-gates are rarely used in practice and usually the fault tree analysis can be completed using only AND and OR-gates. However, later, in the attack tree analysis, Khand argues in [43] that the conventional AND-nodes and OR-nodes of attack trees can not adequately capture all attacks towards those systems and redefines PRIORITY-AND and several new ones. Yager also argues in [88] that the usual attack tree building blocks are insufficient and proposes a probabilistic node that allows modeling of uncertainty in the number of children that needs to be satisfied.

So far, we have been able to do the attack tree analysis using only AND-nodes and OR-nodes and in the following text we are assuming attack trees consisting only of those nodes.

1.2.2 Fault Tree Construction

During the construction of the fault trees, the system analyst should consider *immediate*, *necessary* and *sufficient* causes for the occurrence of the events. The immediate causes are usually not the *root* or *basic* causes for something, but instead, the immediate causes or immediate mechanisms. Without following this rule, the valid fault tree branches may be missed and the valid causes for system failure might not be analyzed.

It can be illustrated by the circuit in Figure 1.2 on the facing page when we analyze the top event "no input signal to D". When we do not consider the passive components (the wires between active components A, B, C, D), the temptation is to list the "no input signal to C" as the cause for our top event. However, when considering *immediate* causes, we should list "no output signal from C" as the cause for our top event. Now we can analyze further the causes to "no output

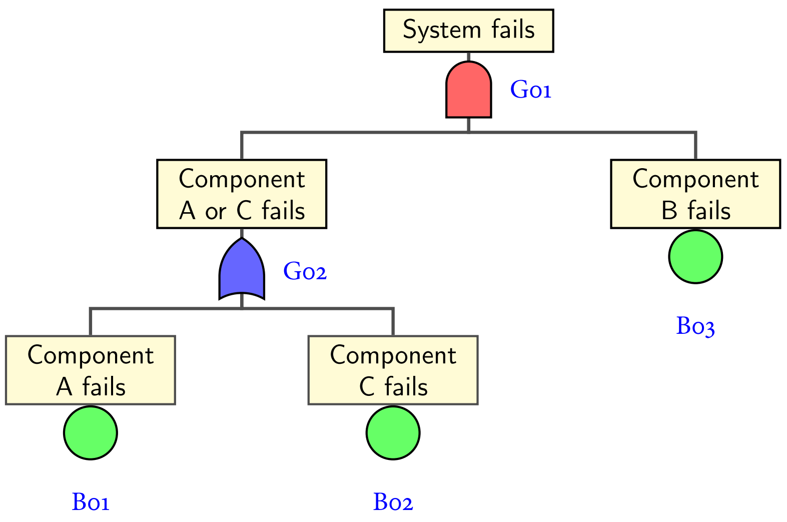


FIGURE 1.1: Sample fault tree for a system with active components A, B, C and with faults $B01$, $B02$, $B03$ and the corresponding Boolean formula $(B01 \vee B02) \& B03$

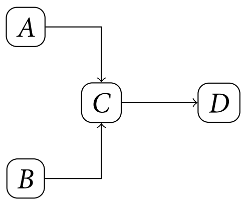


FIGURE 1.2: Sample circuit consisting of active components A, B, C, D

signal from C" as follows:

(1) "There is an input to C, but no output from C" and

(2) "There is no input to C".

Without following this "Immediate causes" rule or "Think Small" rule, we would have missed the (1) event, essentially a "broken C" event.

Additional best practices, which are helpful to the fault tree construction, also apply to the attack trees and therefore we list them here:

1. Verbosity Rule – Describe precisely, what the fault event is and when it oc-

curs. This results in rather verbose event descriptions, so be it.

2. No Miracles Rule – If the normal functioning of the component propagates a fault sequence, then it is assumed that the component functions normally.
3. Complete the Gate Rule – All inputs to a particular gate should be completely defined before further analysis of any of them is undertaken.

Differently from the attack trees, where we do not explicitly analyze intermediate events, the fault trees require also the "No Gate-to-Gate Rule", which says that gate inputs should be proper events and gate output should be a proper event.

Fault Tree Analysis is now specified as a standard [36] and is well-established in engineering and in the next section we will show how similar concepts are applied to the security field.

1.3 Attack Tree Analysis

Some researchers on cyber attacks have pointed out that cyber attacks in the wild are combined from multi-stage attack methods [8, 17, 74] and according to that view, the hierarchical attack modeling approach, which allows description of larger more complicated attacks consisting of multiple smaller attacks, seems to be very appropriate. The idea of the hierarchical security analysis based on the fault tree analysis was applied to the information security first in 1991. The seminal paper [87] was the response to the U.S. Department of Defense requirements [77] for the security engineering process. The solution was called *Threat Logic Trees* and the tree root was the high-level potential threat, which was to be decomposed to the tree nodes, either AND or OR-nodes. Finally, the tree leaves are the external attacker actions which do not require further decomposition.

Unfortunately, the publication by Weiss went without much notice and the subject was picked up much later, first in the year 1998 by Salter *et al.* in [65] and a year later by Schneier in [67, 68]. The methodology is now called *attack tree analysis* and still follows the same idea. The AND-nodes are such tree elements, which are only satisfied if all the child nodes are satisfied as well. The OR-nodes are satisfied when any on the child nodes are satisfied as well. The tree leaves correspond to the simple attacks that are not divided further in the context of that particular attack tree.

For example, consider the example of the attack tree in Figure 1.3 on the next page. The root node is labeled "Obtain administrator privileges", which is the goal

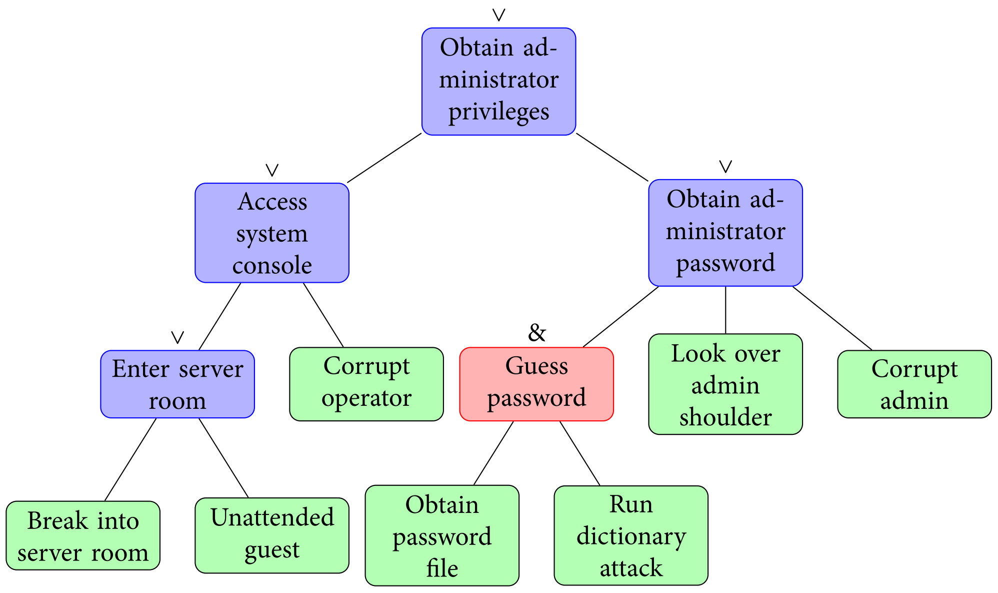


FIGURE 1.3: Example attack tree for attacking the server in the server room (from [87]). The AND-nodes are depicted with red background and with & label above the node, the OR-nodes are depicted with blue background and with ∨ label above the node and attack tree leaves are depicted with green background.

for the attacker. It is further divided into two possible cases and the root node is denoted as OR-node. To achieve the goal of having administrator access to the server, either "Access system console" or "Obtain administrator password" could be pursued. Both branches are further divided, until they represent understandable and simple *elementary attacks*. One possible way to realize this example attack tree is to mount the "Obtain password file" and "Run dictionary attack" attacks. If they are both successful, then the "Guess password" node is realized and in turn, the "Obtain administrator password" node is realized and finally, the "Obtain administrator privileges" root node is realized.

Creating the attack tree structure with the corresponding elementary attack descriptions allows an analyst to argue about the possible attacks and to describe the security situation in a structured, hierarchical way. The resulting attack tree could be easily shared between the security team and studied together. In fact, Steffan and Schumacher even suggest a Web-based method for collaborative attack modeling in [72]. When classifying the elementary attacks into "plausible" and "impossible", the team can informally also argue about the security of the whole system. If there is a corresponding attack suite which consists of plausible elementary attacks, then it would be clear that the attack tree is realizable by an attacker and the system security is lacking.

Guidelines in the scientific literature of how a security analyst should actually create a usable attack tree are scarce. Since fault trees and attack trees are quite similar in the structure, some of the suggestions in Section 1.2.2 on page 20 also apply to attack trees.

1. Verbosity Rule – Describe precisely, what the attack is and when it occurs. If this results in a verbose descriptions, so be it.
2. Complete the Node Rule – All children of the particular node should be defined before further analysis of any of them is undertaken.

Also, in [55] the following top-to-bottom process for creating attack trees for the security protocol analysis is proposed.

1. The root represents the generic ultimate goal of attacking the target security protocol.
2. The second level represents the security properties that the attacker attempts to violate.
3. The third level represents the mechanism exploited by the attacker, as for example, key exchange, message truncation, and so on.
4. Subsequent levels represent attack stages to achieve level 3 sub-goals.

1.3.1 Attack Tree Foundations

To properly argue about the attack trees and the possibility to attack the system, we need a more solid theory. In 2005, Mauw and Oostdijk published their paper [53] which gave the theoretical foundations of the attack trees and the attribute propagation rules. The definitions of the attack tree components are somewhat different from our approach, therefore we do not repeat the original definitions here, but we do cover the important findings of that paper.

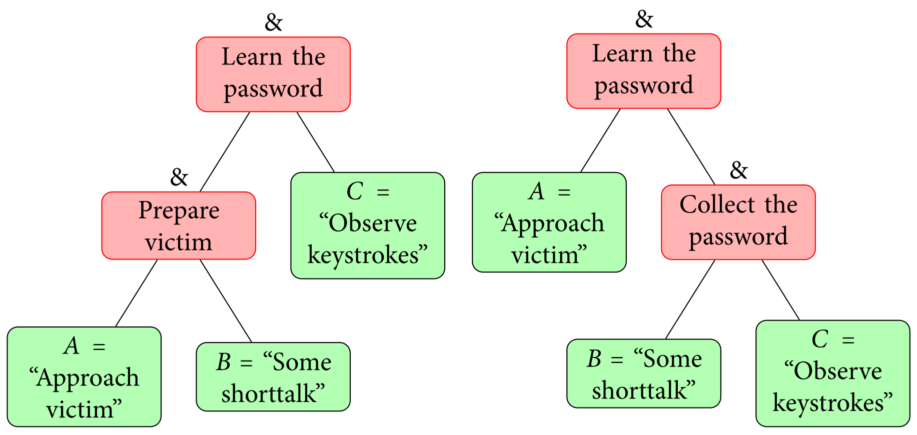


FIGURE 1.4: Example of equivalent attack trees (from [53]). The tree on the left can be described by the Boolean formula $(A\&B)\&C$ and the tree on the right by the formula $A\&(B\&C)$, having the same set of elementary attacks A, B, C

Mauw and Oostdijk considered the following aspects of attack trees and argued that attack tree models have to have formal definitions of:

- how the attack tree is composed from elementary attacks and attack tree nodes,
- semantics of the attack tree itself, i.e. how to decide what that particular attack tree represents or means,
- semantics of attack trees must have the associativity and distributivity properties,
- how to compute the attack tree value by attribute values assigned to the elementary attacks.

When those requirements are fulfilled by the particular attack tree model, then it is a consistent and usable model. For example, when we consider an example attack trees from Figure 1.4, we intuitively understand that they represent the same attacks. In fact, they can be described by the equivalent Boolean formulas $(A\&B)\&C \equiv A\&(B\&C)$, because they are both composed of the same elementary attacks and attack tree semantics should have the associativity property.

If the attack tree model includes some sort of parameter computation rules, the equivalent attack trees should have the same value. Otherwise, the attack tree values depend on the order of attacks and even though different analysts could come up with equivalent attack trees, their results will not be comparable. We will later learn that not all current attack tree models follow that essential rule.

Chapter 2

Quantitative Security Risk Assessment

The Fault Tree Analysis and Attack Tree Analysis create the foundation for security modeling, which we will use in this thesis. The next step in the scientific modeling is to try to invent some metrics and measurements, so that the studied field, information security, could be objectively quantified.

We first refer to [78], where Verendel argues that even though the quantitative risk assessment has been tried for almost 30 years, there is still no validation that security even is measurable. Verendel critically surveys 90 papers of security models, security metrics and security frameworks, from the time period of 1981 until 2008. Only 26 papers included some kind of empirical studies and even then the experiments or observations were usually not repeated, they only included data from the limited time period or focused on one particular case study, and in some papers only applicability of the proposed method was studied. Verendel therefore concludes that quantified security is a *weak hypothesis*, meaning that quantified security models have not been corroborated by predicting outcomes of experiments in repeated large-sample tests.

Even so, possibly without corroboration and perhaps without solid quantified models, the attack tree analysis has already been used for numerous times for studying the security of real-life situations. For example, the protocol security analysis has been performed with attack trees for digital content protection protocols in [33, 34], BGP routing protocol in [15], Network on Wheels inter-vehicle communication systems in [2], and WTLS security protocol used in WAP-enabled mobile phones in [55].

Regarding information systems, the security of notional online banking systems has been analyzed in [25], the cross-site request forgery attacks in [51], consumer information privacy protection taxonomy is developed in [62], and credential repository and certificate authority system for the Globus grid computing toolkit is analyzed in [63]. The security of e-voting systems has been analyzed in [10] and RFID networks in [18].

The security of SCADA systems responsible for controlling power distribution have been studied with attack trees in [60, 73, 13] and security of digital instrumentation and control systems for nuclear reactors in [44].

To help analysts with jobs like that, commercial software packages, such as AttackTree+ (company homepage at `www.isograph-software.com`), PTA Professional Edition (`www.ptatechnologies.com`) and SecurITree (`www.amenaza.com`), have also been developed to help with security analysis, however, little is known what their security models are based on.

What are the current options for quantified security analysis then?

2.1 Quantitative Fault Tree Analysis

First, we will review the results of the fault tree analysis. The main question there is to find out if the system as a whole is likely to fail and what components are the most probable source of the failure. After the dependency graph between the events has been created and after the reliability data of the components have been found out from the experimental results or from the manufacturers' specifications, the following mathematical theory can be applied.

The fault tree with the AND and OR-gates corresponds to the Boolean formula. For that tree, the *minimal cut sets* are found, which are the smallest combinations of component failures which, if they all occur, will cause the top event to occur. In a sense, the Boolean formula is converted to the *disjunctive normal form* (DNF) and the clauses in that form are the minimal cut sets. The probability of the cut set E_i happening can be computed by simply multiplying together the probabilities of the individual faults A_j in that cut set, i.e.

$$\Pr[E_i] = \prod_{A_j \in E_i} \Pr[A_j] \ .$$

We are interested in the probability of a total system failure, the $\Pr[E] = \Pr[\bigcup_E E_i]$. In general, when $E = A + B$, then $\Pr[E] = \Pr[A] + \Pr[B] - \Pr[A \cdot B]$.

If events A and B are independent and the $\Pr[A] < 0.1$ and $\Pr[B] < 0.1$, then $\Pr[A \cdot B] = \Pr[A] \cdot \Pr[B]$ is small compared to $\Pr[A] + \Pr[B]$ and we can use the "Rare Event Approximation" [14]. Indeed, usually the component failure probabilities are rather low (in the order of 10^{-5} or even smaller) and therefore, a simplification can be made that the probability of any cut set happening is

$$\Pr[E] = \Pr\left[\bigcup_E E_i\right] \approx \sum_E \Pr[E_i] \ .$$

The relative importance of some minimal cut set can be found by computing the ratio of $\Pr[E_i] / \Pr[E]$, which allows the analyst to focus on those cut sets which contribute most to the failure of the system and perhaps make some design changes, which will improve the system reliability.

For the Boolean formula with the n number of variables, there can be up to 2^n possible satisfying combinations and also cut sets for the fault tree. Fault tree analysis avoids the exponential problems by only considering such kind of minimal cut sets, which consists of up to k components (k being 2 or 3) and just ignoring such sets, which consist of larger amount of components. Because the probability of the single component failure is small, the probability of failure for cut sets with many components is going to be very small and not relevant for the whole system.

2.2 Combining Fault Tree Analysis and Attack Trees

In [26] an attempt is made to combine threats and vulnerabilities, resulting the so-called n-dimensional view of the attack tree. Later in [28] Fovino *et al.* combine fault-tree analysis and attack trees and propose a quantitative security risk assessment method. Formal definitions of a fault tree and an attack tree are provided and a mathematical model for the calculation of system fault probabilities is presented.

They are using

$$P_{out}AND_i = \prod_{k=1}^{n} P_{in}(k,i) \ , \tag{2.1}$$

the propagation rule for the AND-nodes, where $P_{in}(k,i)$ is the k-th input probability to the i-th AND-node and

$$P_{out}OR_i = 1 - \prod_{k=1}^{n} (1 - P_{in}(k,i)) \ , \tag{2.2}$$

the propagation rule for the OR-nodes, assuming that the inputs to the OR-node are mutually independent.

The fault probabilities are propagated towards the root node and the fault probability of the total system is finally computed. Unfortunately, they are only considering the success probability of attacks and faults and no other economic parameters. Even though they discuss several parameters, such as *attack severity* and *attack plausibility* in [27], the formal model to compute the attacker's gain from the attack tree is not yet developed.

An interesting paper [61] by Ray and Poolsapassit proposes a methodology to monitor insider users and alert the possible malicious attacks. They first create the attack tree with the insider ultimate goal as the root of the tree and then use the known privileges and job descriptions of the inside users to strip the attack tree to only include the relevant portion of the tree for that user, essentially creating minimal cut set for that particular attacker.

The probability of an insider attack is computed at any given time t by considering how far the attacker has progressed in the attack tree. If the attacker has already succeeded with n elementary attacks, but the root node of the attack tree would require the completion of m elementary attacks, the probability of the insider attack is the n/m. To compute the number of elementary attacks m_s required for the attack tree node s, the following formula is used:

$$m_s = \begin{cases} \sum_i^k m_i + k & \text{if } s \text{ is AND-node,} \\ \min_i^k m_i + 1 & \text{if } s \text{ is OR-node,} \end{cases} \tag{2.3}$$

where $s_1, \ldots, s_k$ are the children of the node s.

Ray and Poolsapassit also do not extend their model to include economic parameters and currently their model will just give a crude approximation of the effort attacker needs to spend to achieve the goal, however, the idea of monitoring the attacker capabilities and its progress seems very promising.

2.3 Economic Threat Modeling

Already Weiss thought about using attack trees to measure security, but the model proposed in [87] is not entirely formalized. He included three kinds of metrics with each of the node in the threat logic trees:

1. System Weighted Penalty (SWP) – impact to the system if the node is suc-

Table 2.1: Attack tree metrics propagation rules from the children nodes to the parent node in the Weiss model (in [87])

	AND-node	OR-node
System Weighted Penalty (SWP)	Estimated by analyst	SWP_i of the child with the maximum risk
Level of Adversary Effort (LAE)	$\max_i$ LAE	LAE_i of the child with the maximum risk

cessfully accomplished,

2. Level of Adversary Effort (LAE) – amount of resources required by the adversary to successfully accomplish the attack,

3. Risk – computed by the equation $\mathrm{SWP}^2/\mathrm{LAE}$.

The metrics were to be found out for each leaf node and then propagated towards the tree root. Specific rules for the parent metric computation are given in Table 2.1. The rules however, are not complete. SWP parameter for the parent AND-node is actually estimated by the analyst itself, i.e. the attack tree computation result from the given input parameters depends on the particular computer and objective results are not possible.

A more solid approach was considered by Schechter and Smith in [66], which introduced economic threat modeling on a macro-economic scale. They considered attackers who are targeting multiple victims in parallel, or sequentially and argue that on a large scale, economically it makes sense for victims to share the attack and vulnerability information. They rely on the idea that adversaries are motivated by financial gain. This assumption itself is not obvious, but builds on the work of the Nobel Prize laureate Gary Becker, who gives the model for the expected utility of committing a crime in his seminal paper [6] in 1968. He takes into account the probability of conviction and the utility of the punishment, which is usually a negative number. The expected utility of committing a crime is then the direct income of the crime plus the probability of conviction times the utility of the punishment.

Game theory has been also used to model the network security, by considering the attacker and system owner players of non-cooperative games. However, when multi-stage games are considered (e.g. in [37, 52, 64]), the exponential growth of the corresponding decision tree prohibits the application of models on complicated situations. When just a single-stage game is considered (e.g. in [7]), then the approach is comparable to attack trees and the game theory is indeed an important component of the model there.

2.4 Attack Graphs

A alternative approach to network security analysis, attack graphs does not have the AND-OR-tree structure like attack trees, but all vertexes are of the same type. Usually the vertexes represent attack states and edges represent a possibility for an attacker to move from one attack state to another, although different interpretations have also been proposed. Some of the attack states denote states where the attacker has achieved his goal. Attack graphs are usually machine-generated from network diagrams, which gives us the list of computers, firewalls, servers and other equipment and connectivity information of them and from the vulnerability databases, which give us information about the attacker possibilities. From vulnerability information and server configuration data, it then becomes possible to generate attack graphs, where each path is a series of exploits that may or may not finally lead to attacker success state. To answer this questions, graph-theoretic analysis is applied, such as the shortest path analysis.

Attack graphs were formally specified by Sheyner *et al.* in [71] in 2002. They relied on the model checking and built the attack graph backwards from the attacker success states, which lead to space savings. However, concepts similar to the attack graphs were already used as early as 1994 in [16] by Dacier and later by Phillips and Swiler in [59].

Since then, the attack graphs theory has been advanced to include quantitative analysis with probabilistic and other metrics, taking into account existing security measures and vulnerability characteristics in [85, 46] and analysis to find the weakest attacker, still capable of successfully attacking your network in [58]. Bayesian analysis is applied to attack graphs in [29, 19]. After the attack graph has been analyzed, additional security measures can be applied in the most effective places, as suggested in [83, 35].

The algorithmic complexity of the attack graph analysis is $\mathcal{O}(V + E)$, where

V is the number of vertexes and E is the number of edges in the attack graph, however, the complexity of attack graph generation is $\mathcal{O}(v^n n!)$, where v is the number of vulnerabilities and n is the number of hosts in the network. This means that in general, attack graphs suffer from the exponential complexity explosion when analyzing practical size networks (see [46, section 5] and [71, section 3.4]).

By making the *monotonicity* assumption on the attack graph analysis, the exponential complexity could be reduced to polynomial complexity. This approach is studied by Ammann *et al.* in [3]. Monotonicity property is interpreted in the attack graph theory to indicate that no action taken by the attacker interferes with the attacker's ability to take any other action, i.e. the attacker never reduces his acquired capabilities. The complexity is even further reduced by Wang *et al.* in [84, 86], where relational views based on attack graphs and corresponding SQL query language is developed, to allow interactive analysis of the subset of a larger network, which is relevant for the current problem in hand.

Even though security analysis based on attack graphs is promising, especially because the analysis can be mostly performed automatically, using the gathered information from networks, we suggest that attack trees can be applied more successfully in a situations were the security of a new information system which is still under development must be analyzed and the appropriate security measures have to be chosen. A security analyst has more freedom to model attacker behavior and to identify new kinds of attacks and perform probabilistic economic analysis, not only rely on the known and enumerated system vulnerabilities.

2.5 Quantified Analysis Approach by Edge *et al.*

Edge *et al.* developed a quantified security analysis methodology based on the attack trees in [23, 24] and later showed the application of the method in [25, 18]. Combining attack trees with protection trees, the approach allows the total cost of the attack and the total cost of protection measures to be computed. They propose the *probability of success*, *cost to attack*, *impact to system* and *risk* as the metrics for each attack tree leaf. Impact is a number from 1 – 10 and describes how much of the attacker's goal is achieved, when the attack realizes, or how much impact is generated to the system. The risk is computed from the other three parameters using the formula

$$\text{risk} = (\text{probability}/\text{cost}) \times \text{impact}\,.$$

Table 2.2: Metrics propagation rules from the children nodes to the parent node according to the attack tree model by Edge *et al.* (in [23, 24])

	AND-node	OR-node
Probability	$\prod_i^n \mathrm{prob}_i$	$1 - \prod_i^n (1 - \mathrm{prob}_i)$
Cost	$\sum_i^n \mathrm{cost}_i$	$(\sum_i^n \mathrm{prob}_i \times \mathrm{cost}_i) / \sum_i^n \mathrm{prob}_i$
Impact	$(10^n - \prod_i^n (10 - \mathrm{impact}_i)) / 10^{(n-1)}$	$\max_i^n \mathrm{impact}_i$

To compute the cost of the whole attack tree, they are giving the rule set in Table 2.2. The propagation rules make sense for the probability parameter and they are equivalent with formula (3.3) of our parallel attack tree model and with formulae (2.1–2.2) of the model by Fovino *et al.*, however, the propagation rules for the attack cost parameter does not make sense. Edge *et al.* give the following rationalization in [24]:

> For an OR node, it is unknown which way an attacker will exploit the system, so the cost is calculated as a weighted average of the child node costs based on the probability of success. An alternate method of calculation uses the lowest cost child node as this would be the worst case for the defender of the system. The weighted average cost calculation retains more information about the likely cost to an attacker.

This conflicts with our interpretation of the attack tree semantics. In our models, the elementary attack parameter Expenses is associated with the preparation costs of the attack and also expected penalties of the attack (the negative utility of the crime in [6]). Those expenses must be spent when the attacker launches some attack suite, irrespective of whether the individual elementary attacks succeed or not. Yes, the attacker can consider different attack suites to satisfy the attack tree and we do not ultimately know, which attack suite will be used, however, the individual Expenses of the elementary attacks must be summed to find out the total cost of the attack suite.

Additionally, the model is not consistent in the sense of the requirements of Mauw and Oostdijk. For example, if we take the two equivalent attack trees $\mathcal{T}_1 =$

Table 2.3: Total cost of attack in the attack trees $\mathcal{T}_1 = (A \vee B) \vee C$ and $\mathcal{T}_2 = A \vee (B \vee C)$ according to the quantified attack tree model by Edge *et al.*

	A	B	C	$(A \vee B)$	$(A \vee B) \vee C$	$(B \vee C)$	$A \vee (B \vee C)$
Probability	0.8	0.8	0.8	0.92	0.99	0.96	0.99
Cost	4	5	6	4.43	5.21	5.6	4.87

$(A \vee B) \vee C$ and $\mathcal{T}_2 = A \vee (B \vee C)$ and compute the total cost of the attack, we get different results, as depicted in Table 2.3. The total cost of tree $\mathcal{T}_1$ is 5.21 units, but the total cost of the $\mathcal{T}_2$ is 4.87 units. The model does not have the associativity property.

This means that the attack tree model outcome depends on how the analyst creates the tree in the first place and as we saw, it even depends on which order the attacks are put to the tree. In our view, this greatly limits the reliability and usability of the model.

2.6 Survivability Analysis Approach by Fung *et al.*

In 2005, Fung *et al.* presented an initial case study in [31] for adopting the attack tree analysis methodology for a survivability study. They create an attack tree to describe the attacker's goal to compromise the confidentiality, integrity or availability of the command and the control information system. From the attack tree, they generate the *intrusion scenarios* (in our models these correspond to *attack suites*) and then compute the attacker cost function to execute those scenarios. The cost function is defined as

$$F_m = \sum_{i=1}^{I_m} D_{m_i} , \tag{2.4}$$

where the D_{m_i} is the level of attack difficulty of leaf nodes listed in the m-th intrusion scenario. The quantitative measure of system survivability is represented by

$$F = \min_{m \in [1,M]} F_m = \min_{m \in [1,M]} \sum_{i=1}^{I_m} D_{m_i} , \tag{2.5}$$

where $m \in [1, M]$ denotes all intrusion scenarios. The underlying assumption of Fung *et al.* is that the system survivability is determined by the weakest link among all intrusion scenarios.

The model is consistent in the sense of formal requirements by Mauw and Oostdijk, however, the model only works on a single parameter, the *attack difficulty*. According to Becker, this alone does not describe the attacker's motivation adequately and multiple parameters should be taken into account.

2.7 Quantified Analysis Approach by Buldas *et al.*

In 2006, Buldas, Laud, Priisalu, Saarepera, and Willemson published their paper [11], in which they build from the assumption of an attacker thinking in rational and economical terms. Otherwise, the structure of the attack trees and the usage of AND-nodes and OR-nodes to construct the tree was unchanged. We will refer to this model in the following text as the *multi-parameter attack tree model*. Even though the other, later proposed models also work with multiple parameters, they usually have more specific distinguishing names.

The model proposed by Buldas *et al.* can be regarded as one the first to apply the model of expected utility of committing a crime by Becker in [6]. Buldas *et al.* started the analysis from the attacker viewpoint and assumed that the attacker and the attacked system will behave according to the following game:

1. The attacker prepares the attack and by doing so, some resources, such as money or time are spent. This is described by the parameter Cost.
2. The attack is then launched. The attacked system has some protective security measures deployed, such as physical locks, firewalls, user authentication, etc. Depending on how strong the security measures are and how much of the resources are used in the preparation phase, the attack will succeed or fail and they consider this as a random event with success probability of p and failure probability of $1 - p$.
3. Systems also usually have detective security measures installed, such as security cameras, access logs or periodic audits. Those measures help to detect if there has been a successful or failed attack and they might lead to the capture of the attacker. The capture means that the attacker needs to pay some penalty. In case the attack itself was successful, we use the expected

Table 2.4: Different outcomes of the attacker game in the attack tree model by Buldas *et al.* (in [11])

Attack status	Attacker status	Outcome of the game
failed	not caught	$-\mathsf{Cost}$
failed	caught	$-\mathsf{Cost} - \mathsf{Penalties}^-$
succeeded	not caught	$\mathsf{Gains} - \mathsf{Cost}$
succeeded	caught	$\mathsf{Gains} - \mathsf{Cost} - \mathsf{Penalties}^+$

penalty π_i^+ and in case the attack failed, we use the expected penalty π_i^-.

4. If the attacker is successful, he will get the price of the game, Gains and this is the main reason why the attacker decides to attack us in the first place.

In this game there are four different outcomes for the attacker, and defining them now is very intuitive. The list of them is given in Table 2.4. It is presumed that the Gains of the game are not confiscated after the attacker is caught and rather that the expected penalty π_i^+ is usually sufficiently large. By extending this game to a larger attack tree, they presume that it is possible to compute the Outcome value for each attack tree leaf, i.e. for each elementary attack and the attacker game still holds for the whole tree as well. When the attacker goal is described by the larger tree, the price of the game is received only after the achievement of the root node.

After (Cost_i, p_i, π_i^+, π_i^-) parameters for all the elementary attacks are estimated, the following computation formulae are used to propagate the parameters to the parent nodes in the attack tree. Additionally, there is the attack tree global parameter Gains, which describes the benefit of the attacker, in case the root node of the attack tree is achieved.

For an OR-node with child nodes ($i = 1, 2$) with parameters (Cost_i, p_i, π_i^+, π_i^-), the parameters $(\mathsf{Cost}, p, \pi^+, \pi^-)$ of the parent node are computed as

$$(\mathsf{Cost}, p, \pi^+, \pi^-) = \begin{cases} (\mathsf{Cost}_1, p_1, \pi_1^+, \pi_1^-) & \text{if } \mathsf{Outcome}_1 > \mathsf{Outcome}_2 \\ (\mathsf{Cost}_2, p_2, \pi_2^+, \pi_2^-) & \text{if } \mathsf{Outcome}_1 \leq \mathsf{Outcome}_2 \end{cases}, \tag{2.6}$$

$$\mathsf{Outcome}_i = p_i \cdot \mathsf{Gains} - \mathsf{Cost}_i - p_i \cdot \pi_i^+ - (1 - p_i) \cdot \pi_i^- . \tag{2.7}$$

Table 2.5: Metrics propagation rules from the children nodes to the parent node according to the attack tree model by Buldas *et al.* (in [11])

	AND-node	OR-node
Probability	$\prod_i^n p_i$	p_i of the child node, which has maximal $\mathsf{Outcome}_i$
Expenses	$\sum_i^n \mathsf{Expenses}_i$	$\mathsf{Expenses}_i$ of the child node, which has maximal $\mathsf{Outcome}_i$

For an AND-node with child nodes with parameters $(\mathsf{Cost}_i, p_i, \pi_i^+, \pi_i^-)$ $(i = 1, 2)$, the parameters $(\mathsf{Cost}, p, \pi^+, \pi^-)$ are computed as follows:

$$\begin{aligned}\mathsf{Cost} = \mathsf{Cost}_1 + \mathsf{Cost}_2\,, \quad p = p_1 \cdot p_2\,, \quad \pi^+ = \pi_1^+ + \pi_2^+\,,\\ \pi^- = \frac{p_1(1-p_2)(\pi_1^+ + \pi_2^-) + (1-p_1)p_2(\pi_1^- + \pi_2^+)}{1-p_1p_2} + \\ + \frac{(1-p_1)(1-p_2)(\pi_1^- + \pi_2^-)}{1-p_1p_2}\,. \end{aligned} \tag{2.8}$$

The formula (2.8) for π^- represents the average penalty of an attacker, assuming that at least one of the two child-attacks was not successful. It is possible to denote all of the expected expenses associated with the elementary attack i as

$$\mathsf{Expenses}_i = \mathsf{Cost}_i + p_i \cdot \pi_i^+ + (1-p_i) \cdot \pi_i^-\,. \tag{2.9}$$

Then it is easy to see that in an AND-node the equality $\mathsf{Expenses} = \mathsf{Expenses}_1 + \mathsf{Expenses}_2$ holds. We give the summary of propagation rules in Table 2.5.

After the parameters p and $\mathsf{Expenses}$ are propagated to the root node R of the attack tree, the value of the total game to the attacker can be computed by

$$\mathsf{Outcome}_R = p_R \cdot \mathsf{Gains} - \mathsf{Expenses}_R\,. \tag{2.10}$$

If the $\mathsf{Outcome}_R > 0$, then it could be said that the system is insecure and the attacker can probably make a profit by launching the attack. If the $\mathsf{Outcome}_R \leq 0$, then it may be that the system is secure, or that we have missed some particular attack tree branches or we have not estimated the elementary attack parameters correctly. In any case, with this kind of multi-parameter attack tree model, the analyst can start to answer questions about how secure their system is and can give economically quantified answers.

2.8 Shortcomings of the Current State of the Art

The attack tree models outlined have several shortcomings. They all (except the model by Fovino *et al.*) use the node parameter propagation from child nodes to the parent nodes. This propagation process in the OR nodes relies on the local optimum decisions, to choose which child node parameters to carry to the parent node. Therefore, the computed utility value is not always the global maximum and in some cases the model might not find the best attack suite. Additionally, some of the models are not consistent with the attack tree foundations by Mauw and Oostdijk, referred to in Section 1.3.1 on page 24, i.e. when the attack tree is transformed to the equivalent tree, the utility value changes.

2.8.1 Global Gains Problem

The price of the attacker game must be specified and in the model by Buldas *et al.*, they are using a global parameter Gains for the whole attack tree. However, when computing the utility of elementary attack, the same global Gains is used, even though by successfully completing a single elementary attack, the attacker does not receive the payoff of the game. Only after succeeding with the root node, the $\mathsf{Outcome}_R$ computation formula (2.7) makes sense, but before that, using the same information to make decisions in the OR-nodes is not actually well-defined.

The models by Weiss and by Edge *et al.* approach this problem by using the *System Weighted Penalty* or *impact* elementary attack parameters. They are essentially dividing the global Gains between the smaller attacks and defining how much damage that particular attack does to the whole system. This solution is still not valid semantically, as the attack tree is only realized when the root node is realized.

2.8.2 Local Optimum Problem

Attack tree models by Weiss, Edge *et al.*, and Buldas *et al.* use parameter propagation rules to compute the utility value of the attack tree root node. We argue

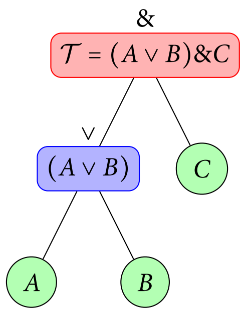


FIGURE 2.1: Attack tree $\mathcal{T} = (A \vee B)\&C$ with attack suites $\{A, C\}$, $\{B, C\}$, and $\{A, B, C\}$, all satisfying the attack tree

that this kind of computation rules does not consider that the attacker has many options to achieve the root goal, i.e., there are many attack suites consisting of elementary attack combinations that will realize the attack tree. Propagation rules try to account for that by using some clever formulae to compute parent OR-node parameters from children parameters, but by doing so, they will need to do local optimization decisions. As we will see, this might not give globally best solutions.

For example, consider the attack tree $\mathcal{T} = (A \vee B)\&C$ in Figure 2.1, with the following parameter values:

$$\begin{aligned} \mathsf{Gains} &= 100\,, \\ p_A = p_B &= 0.8\,, \\ p_C &= 0.9\,, \\ \mathsf{Expenses}_A = \mathsf{Expenses}_C &= 11\,, \\ \mathsf{Expenses}_B &= 10\,. \end{aligned}$$

In the node $(A \vee B)$ formula (2.6) is applied and using that limited local information, it makes sense to choose the attack A. The computation routine is as follows:

$$\begin{aligned} \mathsf{Expenses}_T = \mathsf{Expenses}_A + \mathsf{Expenses}_C = 11 + 10 &= 21\,, \\ p_T = p_A \cdot p_C = 0.8 \cdot 0.9 &= 0.72\,, \\ \mathsf{Outcome}_T = 100 \cdot 0.71 - 21 &= 51\,. \end{aligned}$$

However, the attack tree semantics does not prohibit launching more than one attack from the OR-node children nodes. In some cases this might be beneficial to

TABLE 2.6: Computation example in the attack tree $\mathcal{T} = (A \vee B)\&C$ with $\mathsf{Gains} = 100$ according to the quantified attack tree model by Buldas *et al.*

	A	B	C	$(A \vee B)$	$(A \vee B)\&C$	$\{A, C\}$	$\{B, C\}$	$\{A, B, C\}$
p	0.8	0.8	0.9	0.8	0.72	0.72	0.72	0.864
Expenses	11	10	11	10	21	22	21	32
Outcome	69	70	79	70	51	50	51	54.4

the attacker if the attacks are relatively cheap to try and by trying multiple attacks, the overall success probability will be higher. In the current case, when trying both attacks A and B, the probability for success becomes $p_A \cdot p_B \cdot p_C + p_A \cdot p_C(1 - p_B) + p_B \cdot p_C(1 - p_A)$ and it equals 0.864, which is significantly greater than 0.72. When computing the outcome for the attack suite $\{A, B, C\}$, we get

$$\mathsf{Outcome}_T = 100 \cdot 0.864 - 32 = 54.4\,.$$

By considering the satisfying attack suites, instead of the propagation rules, and choosing the attack suite, which yields the maximal utility, we will sometimes get better expected outcome values. This error becomes especially bad when the model computes the attacker expected outcome being negative and in case there exists a global maximum, which provides the positive expected outcome. In this case we might have a false impression of the security of our information system.

2.8.3 Tree Transformations

We gave examples of equivalent attack trees already in Section 1.3.1 on page 24 and in Figure 1.4 on page 25. The counter-example in Section 2.5 on page 33 showed in Table 2.3 that Edge *et al.* model does not have the associativity property.

Mauw and Oostdijk also specify that attack tree models should have the distributivity property, e.g. the attack trees in Figure 2.2 on the following page should be equivalent because we could take the attack tree $\mathcal{T}_1 = A \vee (B\&C)$ and apply, for example, De Morgan laws to it and get the equivalent Boolean formula $(A \vee B)\&(A \vee C)$, which describes a tree $\mathcal{T}_2$. The utility values of those trees should be equal.

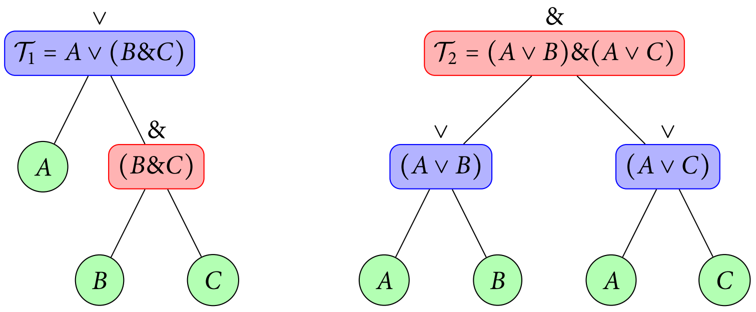


FIGURE 2.2: Attack tree $\mathcal{T}_1 = A \vee (B\&C)$ and the equivalent tree $\mathcal{T}_2 = (A \vee B)\&(A \vee C)$

However, by applying such transformations to formulae, we will introduce duplicated elementary attacks to them. Because such Boolean formulae cannot be represented as trees, we cannot really apply the tree propagation rules there and we cannot compute the tree utility values. This alone should indicate that the current attack tree models are not consistent and the utility value depends on the particular structure of the attack tree.

As different analysts may build different attack trees for the same situation, even though, they may be equivalent in the sense of corresponding Boolean formulae, the outcome will be different and the result of the attack tree analysis cannot be trusted.

Chapter 3

Parallel Attack Tree Model

The multi-parameter attack tree model by Buldas *et al.* is not consistent with Mauw and Oostdijk's work [53]. Equivalent attack trees should result in the same outcome values and therefore we have introduced a new model [39], which is consistent and provides greater outcome values than the original multi-parameter attack tree model.

In this chapter we will describe the model and provide examples of cases with greater outcome values. The downside is that the outcome computation routine is much more complex and therefore we discuss several optimizations and approximations, which will still allow work with reasonable size attack trees.

3.1 Formal Definitions

We are using the propositional directed acyclic graph (PDAG) definition from [81], with the simplification that we do not consider the $\neg$ function. Even though we are strictly talking about directed acyclic graphs where elementary attacks can occur in many branches, we are still calling them trees.

Definition 3.1: *Elementary attack is the lowest level of abstraction of attacks, which do not have any internal structure within the scope of the particular attack tree. Elementary attacks are the leaves of the attack tree.*

Definition 3.2: *Attack tree $\mathcal{T}$ is a simplified PDAG structure $(V = N \cup X, n_0, E)$, of the following elements:*

1. *the set of leaves $X = \{X_1, \ldots, X_n\}$ represents the elementary attacks, which*

are considered as propositional variables having values of true *or* false, *correspondingly, if the elementary attack has been tried and was successful or has been tried and failed,*

2. *the set of nodes* $N = \{N_1, \ldots, N_m\}$ *represents the logical functions of either* & *and* $\vee$. *The function* & *evaluates to* true *if all of its children evaluate to* true *and function* $\vee$ *evaluates to* true, *if some of its children evaluate to* true,
3. $n_0 \in N$ *is the root node of the PDAG, which does not have any parents,*
4. $E = \{(a, b) : a \in V \text{ and } b \in N\}$ *is the set of directed edges between leaves X and nodes N or between nodes N themselves.*

Definition 3.3: *Attack suite* $S \subseteq X$ *is the set of elementary attacks, which have been chosen by the attacker to be launched and used to try to achieve the attacker goal.*

Definition 3.4: *We say that the attack tree* $\mathcal{T}$ *is satisfied by the attack suite S and the goal of the attacker is achieved if the Boolean function corresponding to the root node* n_0 *evaluates to* true *when all elementary attacks from the attack suite S have been tried and they have been evaluated to* true *and* false *values, correspondingly, if the elementary attack was successful or failed.*

Note that because we are considering only monotonic Boolean formulas, the trivial assignment $X_1 :=$ true, $\ldots$, $X_n :=$ true always evaluates $\mathcal{F}$ to true.

3.2 Attack Tree Parameters

We still follow the same basic game-theoretic approach of the original multi-parameter attack tree model and we presume that the attacker makes decisions in each elementary node according to the following game.

1. The attacker has to spend Cost_i resources to prepare and launch the elementary attack.
2. With the probability p_i the attack will succeed and with probability $1 - p_i$ the attack will fail.
3. Depending on the detective security measures, the attacker sometimes has to carry additional costs after failing or succeeding with the attack. Adding

them to the preparation costs, we get the general $\mathsf{Expenses}_i$ parameter for each elementary attack.

Additionally, we still have the global parameter Gains for the whole attack tree, which describes the benefit of the attacker, in case the root node is realized. The attack tree itself is composed of the same kind of AND-nodes and OR-nodes and in case the attacker is able to successfully carry out the necessary attacks, which satisfy the corresponding Boole formula, then we say that the attack tree is realized and the attacker is successful.

For the whole attack tree, the attacker game can be described as follows:

1. The attacker constructs the attack tree with AND-nodes and OR-nodes and evaluates the parameters of the elementary attacks.
2. The attacker considers all potential *attack suites*, i.e. subsets $S \subseteq X$, where $X = \{X_1, \dots, X_n\}$. Some of them satisfy the Boolean formula $\mathcal{F}$ and some do not. For those attack suites which allow the root node to be realized, the attacker computes the outcome value $\mathsf{Outcome}_S$.
3. Finally, the attacker chooses that attack suite which provides the greatest expected outcome and launches the corresponding elementary attacks.

The exact outcome of the attacker can then be written formally as

$$\mathsf{Outcome} = \max\{\mathsf{Outcome}_S : S \subseteq X,\ \mathcal{F}(S := \mathsf{true}) = \mathsf{true}\}\,. \tag{3.1}$$

Here, the notion $\mathcal{F}(S := \mathsf{true}) = \mathsf{true}$ notes the evaluation of the formula $\mathcal{F}$ to value true, when all variables in the attack suite S are assigned the value true and all other the value false, i.e. the attack suite S satisfies the Boolean formula $\mathcal{F}$.

3.3 Outcome Computation

Because we are no longer propagating the node parameters towards the root of the attack tree, but we are instead considering the individual attack suites, the computation formulae are changed as well. The $\mathsf{Outcome}$ value for attack suite S can be computed as:

$$\mathsf{Outcome}_S = p_S \cdot \mathsf{Gains} - \sum_{X_i \in S} \mathsf{Expenses}_i\,. \tag{3.2}$$

When computing the success probability p_S of the attack suite S we must take into account that the suite may still contain redundancy and there may be (proper) subsets $R \subseteq S$ sufficient for materializing the root attack. Because we are using the full suite of S to mount an attack, those elementary attacks in the $S \setminus R$ will contribute to the success probability of p_R with $(1 - p_j)$. Thus, the total success probability can be computed as

$$p_S = \sum_{\substack{R \subseteq S \\ \mathcal{F}(R:=\mathsf{true})=\mathsf{true}}} \prod_{X_i \in R} p_i \prod_{X_j \in S \setminus R} (1 - p_j) \,. \tag{3.3}$$

Using this kind of a computation model, we no longer depend on the actual structure of the attack tree or the Boolean formula $\mathcal{F}$. As long as the same attack suite S satisfies both $\mathcal{F}$ and equivalent $\mathcal{F}'$, the $\mathsf{Outcome}_S$ is still the same. Therefore this model corresponds to the requirements of Mauw and Oostdijk.

THEOREM 3.1: *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two attack trees. If $\mathcal{T}_1 \equiv \mathcal{T}_2$, then* $\mathsf{Outcome}_{\mathcal{T}_1} = \mathsf{Outcome}_{\mathcal{T}_2}$. *The parallel attack tree computing model defined by formulae (3.1 – 3.3) is consistent.*

Proof. The equivalence $\mathcal{T}_1 \equiv \mathcal{T}_2$ means that the corresponding Boolean formulae are equivalent as well and $\mathcal{F}_1 \equiv \mathcal{F}_2$. The formulae (3.1 – 3.3) do not use the particular structure of the attack tree $\mathcal{T}$, they only work on the satisfiability of the corresponding Boolean formula $\mathcal{F}$. Since the equivalent Boolean formulae are satisfied by the same set of assignments, we will have the same attack suites for both trees and $\mathsf{Outcome}_{\mathcal{T}_1} = \mathsf{Outcome}_{\mathcal{T}_2}$. Because Boolean formulae have associativity, commutativity and distributivity properties, the requirements of Mauw and Oostdijk for the attack tree model consistency are also fulfilled. $\square$

An example attack tree computation given in Section 2.8.3 on page 41 also illustrates the current model. This time the attack suites $\{A\}$, $\{B, C\}$ and $\{A, B, C\}$ satisfy both attack trees $\mathcal{T}_1 = A \vee (B \& C)$ and the equivalent tree $\mathcal{T}_2 = (A \vee B) \& (A \vee C)$ (see Figure 2.2 on page 42) and by applying formula (3.2) to the same parameters we get:

$$\begin{aligned}
\mathsf{Outcome}_{\{A\}} &= 0.8 \cdot 10000 - 1100 = 4200\,, \\
\mathsf{Outcome}_{\{B,C\}} &= 0.64 \cdot 10000 - 2200 = 4200\,, \\
\mathsf{Outcome}_{\{A,B,C\}} &= 0.77 \cdot 10000 - 3300 = 4380\,, \\
\mathsf{Outcome}_{\mathcal{T}_1} = \mathsf{Outcome}_{\mathcal{T}_2} &= 4380\,.
\end{aligned}$$

3.4 Comparison with Attack Tree Model by Buldas *et al.*

In Section 2.8.2 on page 39 we already mentioned that the attack tree computation model by Buldas *et al.* does not always find the best attack suite and we presented the counter-example. In this section, we can present more formal proof that the outcome value computed by formula (3.2) is at least as great as the outcome value computed by formulae (2.6–2.8).

Theorem 3.2: *Let us have an attack tree $\mathcal{T}$. Let the parallel model find the best attack suite S and let the best attack suite found by the model described in the current chapter be S and by the model described in Section 2.7 on page 36 be S'. Let the corresponding outcomes (computed using the respective routines) be* $\mathsf{Outcome}_S$ *and* $\mathsf{Outcome}_{S'}$. *The following two claims hold:*

$$\mathsf{Outcome}_S = \mathsf{Outcome}_{S'} \quad \text{if } S = S'\,, \tag{3.4}$$

$$\mathsf{Outcome}_S \geq \mathsf{Outcome}_{S'} \quad \text{if } S \neq S'\,. \tag{3.5}$$

Proof. To prove formula (3.4) we need to show that

$$\mathsf{Outcome}_{S'} = p_S \cdot \mathsf{Gains} - \sum_{X_i \in S} \mathsf{Expenses}_i\,.$$

First note that the attack suite S' is minimal in the sense that none of its proper subsets materializes the root node, because only one child is chosen in every OR-node according to formula (2.6). Hence, $p_S = \prod_{X_i \in S} p_i$. Now consider how the $\mathsf{Outcome}_{S'}$ of the root node is computed by formula (2.7). Let the required parameters of the root node be p', Gains' and $\mathsf{Expenses}'$. Obviously, $\mathsf{Gains}' = \mathsf{Gains}$.

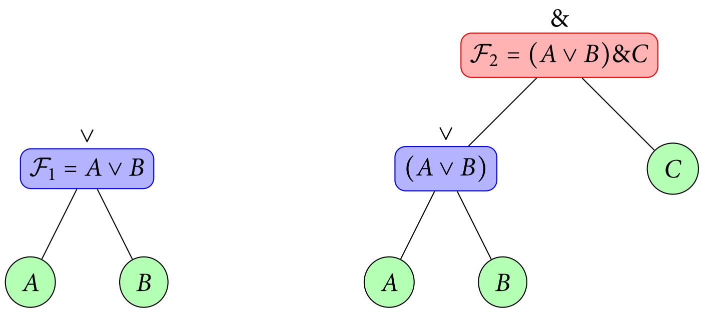


FIGURE 3.1: Attack trees with corresponding Boolean formulae $\mathcal{F}_1 = A \vee B$ and $\mathcal{F}_2 = (A \vee B)\&C$

By looking at how the values of the attack success probability and the expected expenses are propagated throughout the tree, we can also conclude that

$$p' = \prod_{X_i \in S} p_i = p_S \quad \text{and} \quad \mathsf{Expenses}' = \sum_{X_i \in S} \mathsf{Expenses}_i \,,$$

which finishes the first part of the proof.

To show formula (3.5) we consider that the attack suite S' found by the previous model is a satisfying assignment to the attack tree $\mathcal{T}$ or the Boolean formula $\mathcal{F}$ as well. Therefore, it will be considered among the other satisfying assignments as well and therefore we are guaranteed to find at least the same assignment S as well if the $\mathsf{Outcome}_S$ is indeed the greatest outcome. Therefore, the $\mathsf{Outcome}_S$ is not less than the $\mathsf{Outcome}_{S'}$. □

However, the question of why the outcome computed by the parallel attack tree model is at least as great as the outcome value of the previous model needs to be discussed further.

If we reach a different attack suite $S \neq S'$, then the $\mathsf{Outcome}_S$ has to be greater or equal to the $\mathsf{Outcome}_{S'}$. In which cases could it be greater? Let us look at the minimal case, at the attack tree with the formulae $\mathcal{F}_1 = A \vee B$ (see Figure 3.1) and the situation where $\mathsf{Outcome}_{\{A\}} \leq \mathsf{Outcome}_{\{B\}} < \mathsf{Outcome}_{\{A,B\}}$. When we apply formula (3.2) and simplify, we get the following two inequalities:

$$\begin{aligned} &\mathsf{Expenses}_A < \mathsf{Gains} \cdot p_A \cdot (1 - p_B) \quad \text{and} \\ &\mathsf{Expenses}_B < \mathsf{Gains} \cdot p_B \cdot (1 - p_A) \,, \end{aligned}$$

depending on which parts of the original inequality we take. This condition could be interpreted as the expenses of the additional elementary attack must be less than the income from succeeding with the attack and failing with other attacks and we can deduce that it is indeed possible for the parallel model to give greater outcome values than the previous model.

If this could be extended to larger trees, then we would have a simple condition to verify. However, analyzing the slightly larger tree with formula $\mathcal{F} = (A \vee B)\&C$ (see Figure 3.1 on the facing page) and the situation $\mathsf{Outcome}_{\{A,C\}} \leq \mathsf{Outcome}_{\{B,C\}} < \mathsf{Outcome}_{\{A,B,C\}}$, it turns out that we get the following inequality:

$$\mathsf{Expenses}_B < \mathsf{Gains} \cdot p_B \cdot p_C \cdot (1 - p_A) \ .$$

However, because there is already p_C involved in this inequality, the condition is not localized to the single OR-node fragment and with larger trees, this condition will be more complex. Therefore, it is not possible to develop a general verification procedure.

3.5 Optimizations

The brute-force approach to actually generating all 2^n possible subsets from the n elementary attacks, then verifying if they satisfy the Boolean formula $\mathcal{F}$ and then blindly computing the outcome by formulae (3.2–3.3) is very slow. However, we can use the existing results from the Boolean satisfiability theory and we can also come up with simplifying optimizations by our own, as described in [42].

3.5.1 DPLL Algorithm

First, we will modify the well-known satisfiability deciding algorithm [21] by Davis, Logemann, and Loveland. Even though the original DPLL algorithm only verified if the given subset of Boolean variables satisfies the Boolean formula, the idea of branching and backtracking can be applied to systematically generating all possible satisfying subsets as well. We will shown that in Algorithm 3.1 on the next page.

Algorithm 3.1: Finding the satisfying assignments with DPLL algorithm

```
Input:  Boolean CNF-formula $\mathcal{F}$
Input:  subset $S$ of $\mathcal{F}$ variables
Input:  set $A \subseteq (X \setminus S)$
 1: Procedure ProcessSatisfyingAssignments($\mathcal{F}, S, A$)
 2: if $\mathcal{F}$ contains TRUE in every clause then
 3:     Process the assignment $A \cup T$ for every $T \subseteq S$
 4:     return
 5: end if
 6: if $\mathcal{F}$ contains an empty clause or $S = \varnothing$ then
 7:     return
 8: end if
 9: if $\mathcal{F}$ contains a unit clause $\{X\}$, where $X \in S$ then
10:     Let $\mathcal{F}'$ be the formula obtained by setting $X$ = TRUE in $\mathcal{F}$
11:     ProcessSatisfyingAssignments($\mathcal{F}', S \setminus \{X\}, A \cup \{X\}$)
12:     return
13: end if
14: Select a variable $X \in S$
15: Let $\mathcal{F}'$ be the formula obtained by setting $X$ = TRUE in $\mathcal{F}$
16: ProcessSatisfyingAssignments($\mathcal{F}', S \setminus \{X\}, A \cup \{X\}$)
17: Let $\mathcal{F}''$ be the formula obtained by deleting $X$ from $\mathcal{F}$
18: ProcessSatisfyingAssignments($\mathcal{F}'', S \setminus \{X\}, A$)
19: return
```

3.5.2 Withdrawing Hopeless Branches

When we are evaluating the attack suites which satisfy the Boolean formula $\mathcal{F}$, it would be very beneficial if we could drop those suites which will probably not be the global optimum suite and therefore cut of the unnecessary branching. It turns out that we can stop the processing as soon as we find the situation in the attack tree, where some of the AND-node child is evaluated to true and the other child to false. The AND-node itself will be evaluated to false and it turns out that there has to exist another attack suite, which will still evaluate this AND-node to false, but will contain fewer elementary attacks i.e. is cheaper and has the same success probability and which will therefore provide greater outcome and is therefore more likely the global maximum attack suite. Formally, we can prove Theorem 3.3 about this.

Theorem 3.3: *Let $\mathcal{F}$ be a Boolean formula corresponding to the attack tree $\mathcal{T}$ (i.e.*

AND-OR-tree, where all variables occur only once) and let S be its satisfying assignment (i.e. an attack suite). Set all the variables of S to true *and all the other to* false *and evaluate all the internal nodes of T. If some AND-node has children evaluating to* true *as well as children evaluating to* false*, then there exists a satisfying assignment $S' \subset S$ and $S' \neq S$ such that* $\mathsf{Outcome}_{S'} \geq \mathsf{Outcome}_S$.

Proof. Consider an AND-node Y having some children evaluating to true and some evaluating to false. Then the node Y itself also evaluates to false, but the set of variables of the subformula corresponding to Y has a non-empty intersection with S; let this intersection be Q. We claim that we can take $S' = S \setminus Q$. First it is clear that $S' \subset S$ and $S' \neq S$. Note also that S' is a satisfying assignment and hence $S' \neq \emptyset$. To illustrate this situation an example attack tree is given in Figure 3.2.

Now consider the corresponding outcomes:

$$\begin{aligned}
\mathsf{Outcome}_S &= p_S \cdot \mathsf{Gains} - \sum_{X_i \in S} \mathsf{Expenses}_i \,, \\
\mathsf{Outcome}_{S'} &= p_{S'} \cdot \mathsf{Gains} - \sum_{X_i \in S'} \mathsf{Expenses}_i \,.
\end{aligned}$$

Since $S' \subset S$, we have

$$\sum_{X_i \in S} \mathsf{Expenses}_i \geq \sum_{X_i \in S'} \mathsf{Expenses}_i \,,$$

as all the added terms are non-negative.

Now we claim that the equality $p_S = p_{S'}$ holds, which implies the claim of the theorem. Let

$$\begin{aligned}
R_S &= \{R \subseteq S \,:\, \mathcal{F}(R := \mathsf{true}) = \mathsf{true}\} \,, \\
R_{S'} &= \{R' \subseteq S' \,:\, \mathcal{F}(R' := \mathsf{true}) = \mathsf{true}\} \,,
\end{aligned}$$

then by formula (3.3) we have

$$\begin{aligned}
p_S &= \sum_{R \in R_S} \prod_{X_i \in R} p_i \prod_{X_j \in S \setminus R} (1 - p_j) \,, \\
p_{S'} &= \sum_{R' \in R_{S'}} \prod_{X_i \in R'} p_i \prod_{X_j \in S' \setminus R'} (1 - p_j) \,.
\end{aligned}$$

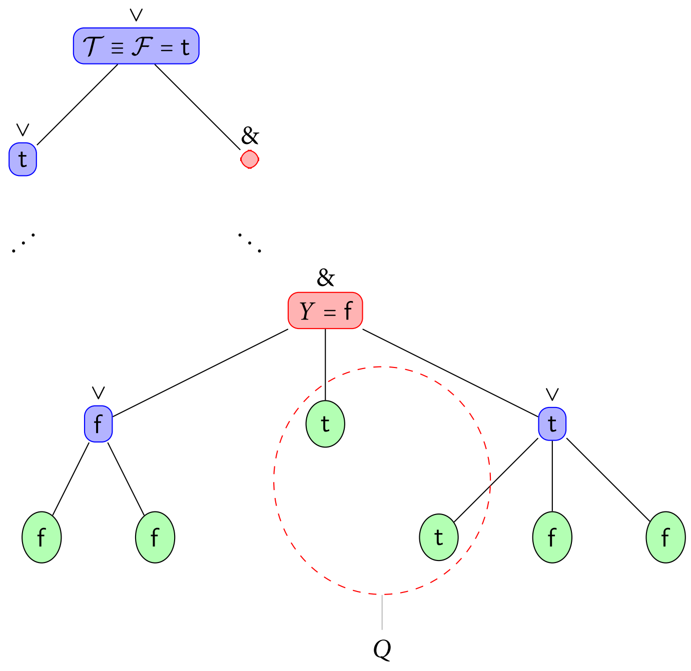


FIGURE 3.2: Illustrating attack tree for Theorem 3.3 on the facing page. The nodes are labelled with the true or false values. The set Q contains the unnecessary leaves from the S set, so that we could take $S' = S \setminus Q$ and the $\mathsf{Outcome}_{S'} \geq \mathsf{Outcome}_S$

We claim that

$$R_S = \{R' \cup Q' \; : \; R' \in R_{S'}, Q' \subseteq Q\}, \tag{3.6}$$

holds, i.e. that all the satisfying subassignments of S can be found by adding all the subsets of Q to all the satisfying subassignments of S'. Indeed, the node Y evaluates to false even if all the variables of Q are true, hence the same holds for every subset of Q due to monotonicity of AND and OR. Thus, if a subassignment of S satisfies the formula $\mathcal{F}$, the variables of Q are of no help and can have arbitrary values. The evaluation true for the root node can only come from the variables of S', proving equation (3.6).

Now we can compute:

$$
\begin{aligned}
p_S &= \sum_{R\in R_S} \prod_{X_i\in R} p_i \prod_{X_j\in S\setminus R}(1-p_j) = \sum_{\substack{R=R'\cup Q'\\ R'\in R_{S'},Q'\subseteq Q}} \prod_{X_i\in R} p_i \prod_{X_j\in S\setminus R}(1-p_j) = \\
&= \sum_{R'\in R_{S'}} \sum_{Q'\subseteq Q} \prod_{X_i\in R'\cup Q'} p_i \prod_{X_j\in S\setminus(R'\cup Q')} (1-p_j) = \\
&= \sum_{R'\in R_{S'}} \sum_{Q'\subseteq Q} \prod_{X_i\in R'} p_i \prod_{X_i\in Q'} p_i \prod_{X_j\in S'\setminus R'}(1-p_j) \prod_{X_j\in Q\setminus Q'}(1-p_j) = \\
&= \sum_{R'\in R_{S'}} \prod_{X_i\in R'} p_i \prod_{X_j\in S'\setminus R'}(1-p_j) \sum_{Q'\subseteq Q} \prod_{X_i\in Q'} p_i \prod_{X_j\in Q\setminus Q'}(1-p_j) = \\
&= \sum_{R'\in R_{S'}} \prod_{X_i\in R'} p_i \prod_{X_j\in S'\setminus R'}(1-p_j) \prod_{X_i\in Q}[p_i + (1-p_i)] = \\
&= \sum_{R'\in R_{S'}} \prod_{X_i\in R'} p_i \prod_{X_j\in S'\setminus R'}(1-p_j) = p_{S'}\,,
\end{aligned}
$$

since $S \setminus (R' \cup Q') = (S' \setminus R')\dot\cup(Q \setminus Q')$. The claim of the theorem now follows easily. □

3.5.3 Efficient Assignments Finding

The regular satisfiability algorithm presents an additional overhead, because we first have to transform the formula $\mathcal{F}$ to the standard CNF notation and we need to process it as a set of clauses, which in turn, are sets of literals. It turns out that by processing $\mathcal{F}$ as a tree, we can apply a more efficient algorithm. We will use ternary logic, i.e. every leaf and node has three possible values, true, false and undefined. The AND, and OR operations on those values are defined in Table 3.1 on the next page.

The new algorithm still follows the basic idea of DPLL branching and backtracking. We start off with the assignment $\{\mathsf{u}, \mathsf{u}, \ldots, \mathsf{u}\}$, meaning that we assign undefined value to each leaf node and then recursively try to assign true and false values to some leaves. When we can evaluate the whole formula $\mathcal{F}$ to false, using the computation rules from Table 3.1 on the following page, we know that we have reached dead end and we will backtrack. When the formula $\mathcal{F}$ evaluates to true, we know that we have reached the group of satisfying assignments of true leaves and undefined leaves can have arbitrary values, therefore we just generate all of the combinations. The rest of the details are given in Algorithm 3.2.

TABLE 3.1: Computation rules for ternary logic

&	t	f	u
t	t	f	u
f	f	f	f
u	u	f	u

∨	t	f	u
t	t	t	t
f	t	f	u
u	t	u	u

ALGORITHM 3.2: Finding the satisfying assignments, optimized version

```
INPUT: Boolean formula $\mathcal{F}$ corresponding to the given AND-OR-tree
 1: PROCEDURE ProcessSatisfyingAssignments($S$)
 2: Evaluate $\mathcal{F}(S)$
 3: IF $\mathcal{F}(S) = \mathsf{f}$ THEN
 4:     RETURN
 5: END IF
 6: IF $\mathcal{F}(S) = \mathsf{t}$ THEN
 7:     Output all the assignments obtained from $S$ by setting all its u-values to t and f in
        all the possible ways
 8:     RETURN
 9: END IF
10: // reaching here we know that $\mathcal{F}(S) = \mathsf{u}$
11: Choose $X_i$ such that $S(X_i) = \mathsf{u}$
12: ProcessSatisfyingAssignments($S/[X_i := \mathsf{f}]$)
13: ProcessSatisfyingAssignments($S/[X_i := \mathsf{t}]$)
```

3.5.4 Computing the Attack Suite Success Probability

Computing the success probability p_S of the attack suite S by formula (3.3) is not very efficient. One basically needs to solve the SAT# problem once more for the set S, to find out if there are subsets $R \subset S$ that still satisfy the formula $\mathcal{F}$. However, it turns out that even though formula (3.3) is good to represent conceptually how the p_S value can be computed, there is a more efficient way.

Even though we cannot use parameter propagation for Outcome computation, we can still use parameter propagation for the success probability. We start with taking $p_i = 0$ for those $X_i \notin S$ and leaving all those $X_i \in S$ with their original

ALGORITHM 3.3: Computing the success probability p_S of the attack suite S

```
INPUT: AND-OR-tree with parameters $p_i$
INPUT: the tree node $m$
INPUT: assignment $S$ of the attacksuite
OUTPUT: $p$ is the success probability of node $m$
 1: PROCEDURE ComputeNodeProbability($m, S$)
 2: IF $X_m$ is leaf node AND $X_m \in S$ THEN
 3:     RETURN $p_m$
 4: END IF
 5: IF $X_m$ is leaf node AND $X_m \notin S$ THEN
 6:     RETURN $0$
 7: END IF
 8: IF $X_m$ is AND-node THEN
 9:     $p := 1$
10:     FOR ALL $i$ child of $X_m$ DO
11:         $p := p \cdot$ ComputeSuiteProbability($i, S$)
12:     END FOR
13:     RETURN $p$
14: END IF
15: IF $X_m$ is OR-node THEN
16:     $p := 1$
17:     FOR ALL $i$ child of $X_m$ DO
18:         $p := p \cdot (1 -$ ComputeSuiteProbability($i, \sigma$)$)$
19:     END FOR
20:     RETURN $1 - p$
21: END IF
```

p_i values. Then we can compute the success probability of the parent node X_m by

$$p_m = \begin{cases} \prod_{j=1}^{k} p_{i_j} & \text{if } X_m \text{ is AND-node,} \\ 1 - \prod_{j=1}^{k}(1 - p_{i_j}) & \text{if } X_m \text{ is OR-node.} \end{cases} \tag{3.7}$$

Recursive Algorithm 3.3 computes the success probability of the attack suite if called as `ComputeNodeProbability(root,` S`)`. The algorithm makes one pass over the structure of the tree, computes the success probability of the root node, which equals p_S, as computed by formula (3.3).

3.5.5 Strategies for Choosing the Branching Point

In the implementation of Algorithm 3.2 on page 54 the programmer has to choose in line 10 which particular yet undefined literal it initializes. One way is to choose just randomly, or in some pre-defined order, but it turns out that by choosing the literals according to some well-defined strategy, we can yet further speed up the attack tree computations.

The main goal is to minimize the number of assignments that will be processed during the recursive branches. If we could find out that this particular branch is not going to result in a satisfying assignment and possible attack suite as early as possible, we could go through the branches more quickly. This is also the reason why we first assign the false value to the undefined literal in line 11, so that we could probably get the false evaluation more quickly and then return to the previous recursion level.

Getting the false evaluation out of the formula $\mathcal{F}$ as quickly as possible seems to depend on the number of AND-nodes, which are on the path from the leaf node up to the root node. The higher the AND-node count, the sooner we should get the false evaluation, because all of the AND-node children must have true values for the AND-node being evaluation to true as well. To verify this hypothesis, we considered random, most-AND and weighted-c-AND strategies.

A random strategy chooses the next literal randomly, most-AND strategy picks the literal, which have the highest number of AND-nodes on the path from leaf to root. The weighted-c-AND strategy prefers AND-nodes, which are closer to the root node. The intuition behind this approach is that when we can exclude a larger subtree, we should be able to cut off more hopeless recursion branches as well, hence it makes more sense to prefer paths with AND-nodes closer to the root. Thus we gave each node on the distance i from the root the weight $1/c^i$, where c is a predefined constant. We experimented with values $c = 2$, $c = 0.5$ and $c = 1$. Note that the most-AND strategy is equivalent to the weighted-c-AND strategy with $c = 1$.

We generated about 6000 random attack trees and computed the outcome for those trees using different strategies and measured the running time for those. The results are given in Figure 3.3 on the facing page. We can see that there is definitely a difference between a random strategy and any weighted-c-AND strategy. However, the differences between the various weighted-c-AND strategies were quite small and almost within the confidence interval with 95% confidence level. We also used the least-squares method to fit the function $a^{-1} \cdot b^n$ to the graphs.

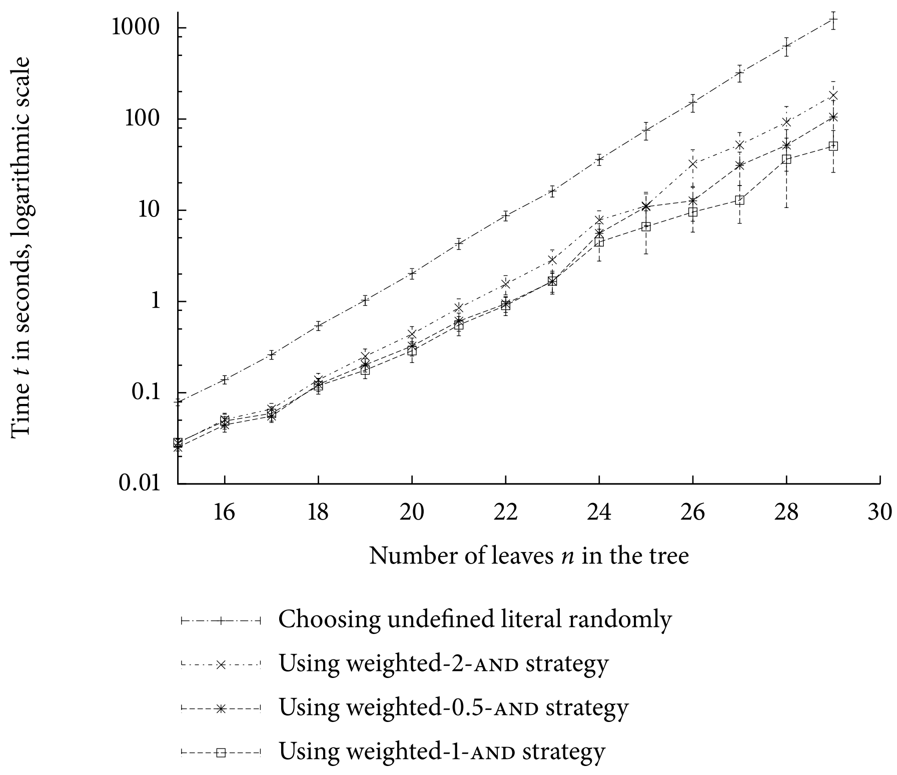


FIGURE 3.3: Performance test results of different strategies for choosing undefined literals. The confidence interval for time t is given with the confidence level 95%. Performance tests were done on 2.33 GHz Pentium Xeon machine.

From this information, we can deduce the approximate complexity of our algorithm implementation in Table 3.2 on the next page.

3.6 Complexity of Parallel Model

Our task is similar to the well known problem of counting the number of solutions to the SAT problem, which is usually denoted by #SAT. Currently the world's best #3SAT problem solving algorithm by Kutzkov in [48] has the worst-case com-

TABLE 3.2: Complexity estimations for different strategies

Strategy	Complexity
Random	$\mathcal{O}(1.90^n)$
Weighted-2-AND	$\mathcal{O}(1.78^n)$
Weighted-1-AND	$\mathcal{O}(1.71^n)$
Weighted-0.5-AND	$\mathcal{O}(1.75^n)$

plexity $\mathcal{O}(1.6423^n)$. As #SAT problems can be converted parsinomically and in polynomial time to the #3SAT problems (see [47], chapter 26), we can roughly compare our algorithm complexity and the #3-SAT problem solver complexity.

Direct comparison is, however, not possible for several reasons. First, our estimate is heuristic and is based on experimental data. Second, we are not only counting all of the possible SAT solutions to the formula $\mathcal{F}$, but we actually have to generate many of them and perform the attacker outcome computations as well. At the same time, we are using the optimization described in Theorem 3.3 on page 50.

Indirect comparison with the result of Kutzkov still shows that our approach works roughly as well as one would expect based on the similarity of our problem setting to #SAT. It remains an open problem to develop more direct comparison methods and to find out whether some of the techniques of #SAT solvers could be adapted to the attack trees directly.

3.7 Genetic Algorithm

Optimizations described in Section 3.5 on page 49 allow us to compute best outcome for the attack tree with 30 elementary attacks in about 1–10 minutes. Attack trees used in practical security analysis tend to be much larger, with hundreds of elementary attacks. Because the proposed computation routines have exponential complexity, processing attack trees as large as hundreds of leaves is not yet possible. This motivates us to search for ways to get approximate results more quickly. One possible option is to use genetic algorithms for finding the optimum attack suite. We refer to the handbook [20] by Davis for an overview of genetic algorithms.

3.7.1 Representing the Solution

To apply genetic algorithms, we will need to decide, how to represent the solution of the attack tree so that we could easily generate the initial random population, then cross the individuals and possibly mutate them and how to sort out the best solutions. Luckily, the current problem falls quite naturally into the required concepts:

- individual – attack suite S represented as a bit array of all the attack tree leaves,
- population – set of attack suites which are currently under consideration,
- fitness function – the outcome value of the attack suite which measures the quality of the individual,
- crossover operation – crossing two attack suites by choosing randomly a parent for each bit vector element
- mutation operation – randomly flipping bits in the individual's bit array.

3.7.2 Generating initial population

Even though the individual concept is rather simple and crossing two individuals to produce descendant is also easy, we will need to decide how to create the initial population of individuals. Simply creating random bit arrays does not guarantee that the resulting individual is satisfying the $\mathcal{F}$ and therefore we might be wasting evolution cycles at the start, until our individuals change to proper solutions. It would be better to start off with all "live" individuals.

To create random satisfying attack suites, we will use Algorithm 3.4 on the following page. We start off from the root of the $\mathcal{T}$ and from AND-node we will choose all children, from OR-node we will choose randomly at least one child. The resulting attack suite will satisfy the $\mathcal{F}$.

3.7.3 Analysis

After this, the implementation of the genetic algorithm is rather straightforward and it is written in Algorithm 3.5 on page 61.

The time complexity of Algorithm 3.5 on page 61 is $\mathcal{O}(gh^2(n + \log h))$, since we need to process g number of generations and it takes about $\mathcal{O}(nh^2)$ to mutate

Algorithm 3.4: Creating randomly satisfying attack suite S for the attack tree $\mathcal{T}$

```
Input: AND-OR-tree fragment $\mathcal{M} \subseteq \mathcal{T}$,
Output: $S$ is the satisfying the formula $\mathcal{F}$
 1: Procedure GenerateAttackSuite($\mathcal{M}$)
 2: if $X_m$ is the leaf node then
 3:     Add $X_m$ to $S$
 4: end if
 5: if $X_m$ is the AND-node then
 6:     for all $i$ of the $M$ children do
 7:         Add set returned by GenerateAttackSuite($i$) to $S$
 8:     end for
 9: end if
10: if $X_m$ is the OR-node then
11:     while $S \neq \varnothing$ do
12:         for all $i$ of the $M$ children do
13:             if random(0,1) = 1 then
14:                 Add set returned by GenerateAttackSuite($i$) to $S$
15:             end if
16:         end for
17:     end while
18: end if
19: return $S$
```

and to verify the liveliness and about $\mathcal{O}(h^2 \log h)$ to sort out the best h individuals from total of $\binom{h}{2} + h$ individuals. When using the algorithm in practice, one needs to decide the specific value for the parameters h and g.

To decide on the reasonable values for those parameters and to verify if the genetic algorithm really finds the global maximum attack suite, we ran experiments on the set of about 6000 randomly generated attack trees. We first computed the exact global maximum attack suite for each tree and then ran the genetic algorithm with various parameters and verified if the correct attack suite was returned.

Running genetic algorithm m times on the attack trees and then comparing the resulting outcome value with the known correct outcome value are random events with binomial distribution $B(m, p)$ of total m tests and with each individual test having success probability of p. The p is essentially the probability that the genetic algorithm returns the correct outcome value. We will need to estimate the confidence interval $(\tilde{p} - \mathrm{CI} \leq p \leq \tilde{p} + \mathrm{CI})$ for the success probability of the genetic algorithm on the attack tree population. Instead of using the stan-

Algorithm 3.5: Finding approximate best attack suite S for the attack tree $\mathcal{T}$ using the genetic algorithm

```
Input: AND-OR-tree $\mathcal{T}$ with estimated parameters,
Input: Size of the population as $h$ and number of generations as $g$
Output: $S$ is approximately best attack suite for attack tree $\mathcal{T}$
 1: Procedure ComputeApproximateAttackSuite()
 2: Generate $h$ individuals attack suites by Algorithm 3.4 on the facing page for the gen-
    eration $g_1$
 3: for $i = 2$ to $g$ do
 4:     Cross $h$ attack suites from previous generation $g_{i-1}$ with each other, producing
        $\binom{h}{2}$ new attack suites
 5:     Mutate each new individual with probability 0.1 and for each of those, flip every
        leaf bit with probability 0.1
 6:     Add $h$ attack suites from the generation $g_{i-1}$ to the current generation $g_i$
 7:     Filter out those individuals, who are live (i.e. $\mathcal{F}(S_j := \mathtt{t}) = \mathtt{t}$))
 8:     Compute the $\mathsf{Outcome}_{S_j}$ for all individuals and choose $h$ best individuals for the
        next generation $g_{i+i}$
 9: end for
10: return Best attack suite $S$ having the largest $\mathsf{Outcome}_S$ from the last generation $g_g$
```

dard binomial confidence interval from [82], which is over-conservative, does not work well with small m values and fails when the p is close to 0 or 1, we are using Agresti-Coull confidence interval from [1], as recommended in the article [50] by Lawrence D. Brown and DasGupta.

We are using 95% confidence level and the corresponding confidence interval

$$\mathrm{CI}_{\mathrm{AC}} = \tilde{p} \pm k\sqrt{\frac{\tilde{p}(1-\tilde{p})}{\tilde{m}}},$$

where $\tilde{p} = \tilde{x}/\tilde{m}$, $\tilde{x} = x + k^2/2$, $\tilde{m} = m + k^2$ and $k = z_{0.05/2} = \Phi^{-1}(1-0.05/2) = 1.96$.

The results are given in Figure 3.4 on the next page for trees up to 29 leaves. It turned out that we were able to get reasonably high level of success probability with parameter functions $h = 2n$ and $g = 2n$, which gave us the overall complexity as $\mathcal{O}(n^4)$. The performance analysis of the genetic algorithm is given in Figure 3.5 on page 63.

This allows us to process attack trees with 70–100 elementary attacks within a reasonable time of 1 to 10 minutes and still find the approximate outcome value. Taking into account that the attack tree parameter estimation process is also not

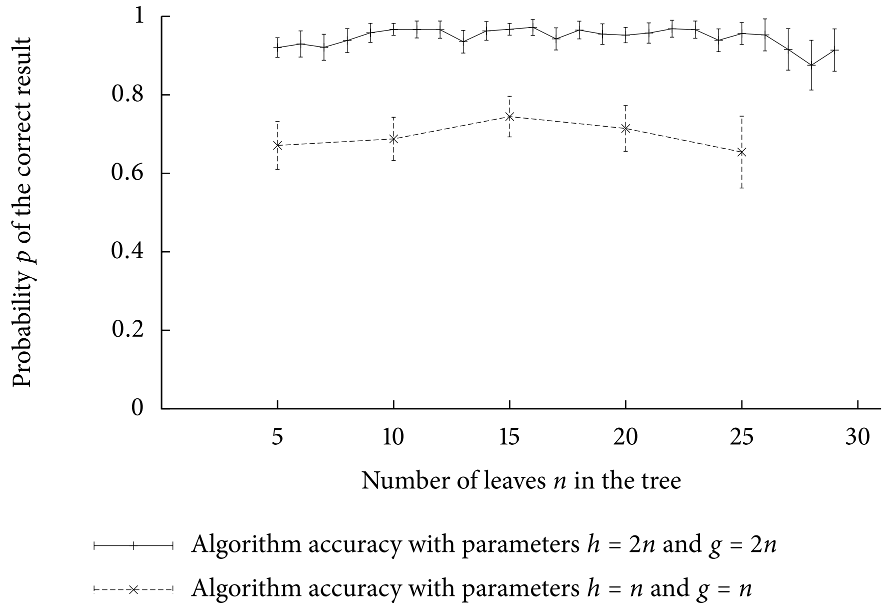


FIGURE 3.4: Accuracy estimation of Algorithm 3.5 on the preceding page for different population size and number of generations values. The confidence interval for probability p is given with the confidence level 95%.

an exact science, we can now use the computed outcome result within some margin of error and use the method in a practical security analysis.

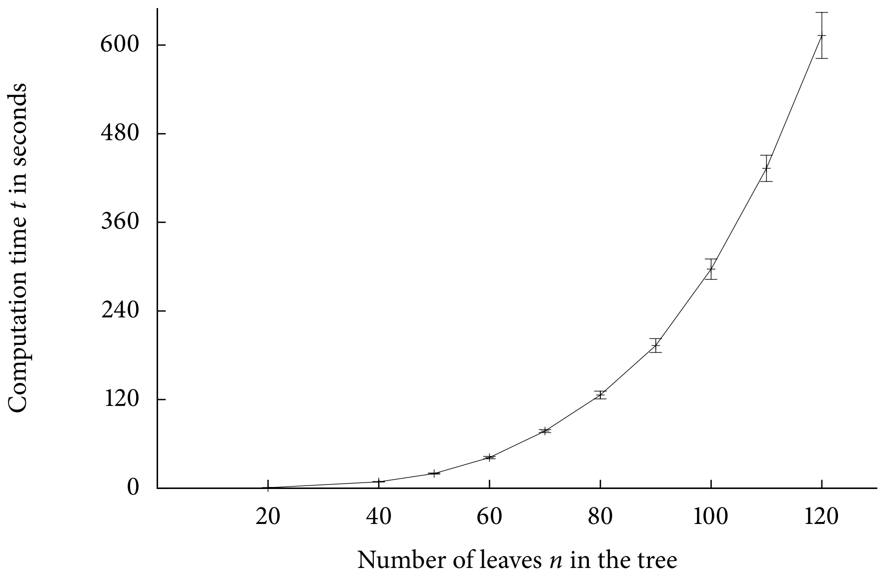


FIGURE 3.5: Performance results of Algorithm 3.5 on page 61 in case of parameters $h = 2n$ and $g = 2n$. The confidence interval for the attack tree computation time t is given with 95% confidence level. Performance tests were done on 2.33 GHz Pentium Xeon machine.

Chapter 4

Serial Attack Tree Model

Attack tree models so far presume that all attacks take place simultaneously and the attacker chooses the best attack suite S before starting the campaign. Parallel model does not allow the attackers to choose different tactics once they have already started. In reality, clever attackers usually try some attacks and if they fail or succeed, then using this additional information, they will choose another approach. By including more information to their decision model, attackers can make smarter decisions and they could get greater expected outcome in the end. Serial model will investigate this situation in more detail.

4.1 Model Description

We will continue to use the basic attacker game, as described in Section 3.2 on page 44 with the regular elementary attack parameters. Elementary attack still has the success probability p and the Expenses parameter for noting the expected cost and penalties for trying the attack. The goal of the attacker is still the root of the AND-OR-tree $\mathcal{T}$ and there is reward of Gains when the corresponding Boolean formula $\mathcal{F}$ is satisfied.

However, in this model we allow the attacker to be adaptive during the execution of the attack suite. The actions of the attacker could be described by the following list:

1. The attacker chooses the $S \subseteq X$, which satisfies the formula $\mathcal{F}$, with $|S| = m$ elementary attacks from the set $X = \{X_1, \ldots, X_n\}$.
2. The attacker will fix the order of the attacks in S by specifying the permu-

tation $\sigma \in \mathfrak{S}_m$.

3. Start launching attacks in the order of $\langle X_{\sigma(1)}, \dots, X_{\sigma(m)} \rangle$.
4. After the attack $X_{\alpha(i)}$ has been launched, the attacker has an option to skip the next attack $X_{\alpha(i+1)}$ if it does not help reaching the attacker goal.
5. The attacker will stop attacking if the goal has already been reached, or if it becomes impossible to fulfill the goal.

The permutation $\sigma : \{1, \dots, m\} \to \{1, \dots, m\}$ specifies the linear ordering relation $\preccurlyeq$ on the set S and we will use the term *ordered attack suite* and notions $\langle S, \preccurlyeq \rangle$ and (S, σ) interchangeably in the following text.

So, for example, if we consider the attack tree from Figure 4.1 on the next page with the ordered attack suite (X, id), the attacker corresponding to our serial model will perform the following actions.

The attacker starts launching attacks from the sequence of $\langle X_1, X_2, X_3, X_4 \rangle$. Independent of whether the first attack X_1 succeeds or fails, there are still other components needed to complete the goal, so attacker tries X_2 as well. If it fails, we see that the whole tree fails, so it does not make sense to try X_3 and X_4. If both X_1 and X_2 have succeeded, we see that it is not necessary to try X_3, since X_1 and X_3 have a common OR-parent, so success or failure of X_4 determines the final outcome. If X_1 fails and X_2 succeeds, we need the success of both X_3 and X_4 to complete the task; if one of them fails, we stop and accept the failure.

The attacker model can be written in a more formal way in Algorithm 4.1 on page 68.

The outcome of the ordered attack suite (S, σ) will be defined as

$$\mathsf{Outcome}_{S,\sigma} = p_\sigma \cdot \mathsf{Gains} - \sum_{X_i \in S} p_{\sigma,i} \cdot \mathsf{Expenses}_i \,, \tag{4.1}$$

where p_σ is the probability that $\mathcal{F}(S) = \mathsf{true}$, i.e., the attacker succeeds with the attack tree goal and $p_{\sigma,i}$ denotes the probability that the node X_i is used during the execution of Algorithm 4.1 on page 68.

The serial model does indeed allow the attacker to make some reasonable decisions during the execution of the ordered attack suite and therefore, it should model the real life more exactly. However, to actually decide which of the subsets S and permutations σ would provide the greatest expected outcome, one has to consider all possible combinations of them. There could be up to 2^n number of subsets and there are $n!$ number of possible permutations for n elements and

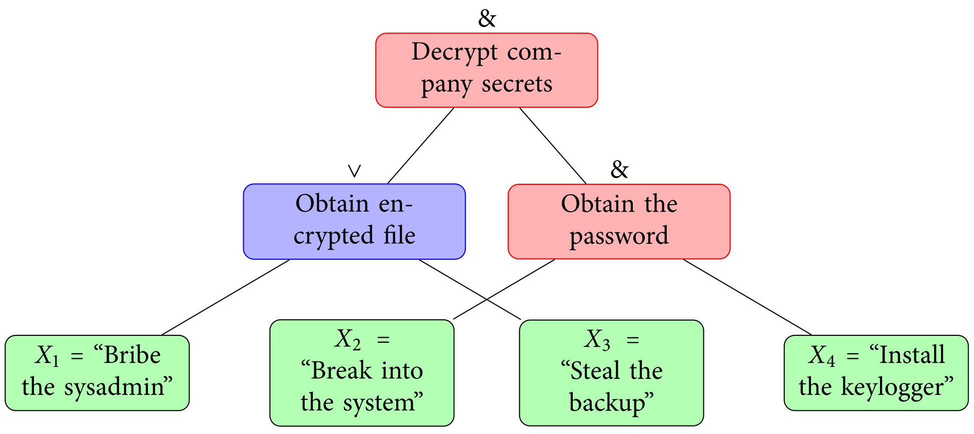

&
Decrypt company secrets
∨
Obtain encrypted file
&
Obtain the password
X_1 = "Bribe the sysadmin"
X_2 = "Break into the system"
X_3 = "Steal the backup"
X_4 = "Install the keylogger"

FIGURE 4.1: An example ordered attack tree $(X_1 \vee X_3) \& (X_2 \& X_4)$. The left-to-right ordering of the leaf nodes in the tree represents the permutation $id = \langle 1, 2, 3, 4 \rangle$ of the $X = \{X_1, X_2, X_3, X_4\}$.

additionally, just computing the $\mathsf{Outcome}_{S,\sigma}$ is also not a trivial task. We will continue with more efficient computation routines in Section 4.3 on page 70 and Section 4.5 on page 73, but first we will compare this model to previous models.

4.2 Comparison with Parallel Attack Tree Model

To compare this model with the previous attack tree models, we first prove that the expected outcome does not depend on the specific form of the attack tree $\mathcal{T}$.

THEOREM 4.1: *Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be two monotone Boolean formulae such that $\mathcal{F}_1 \equiv \mathcal{F}_2$, and let $\mathsf{Outcome}^1_\sigma$ and $\mathsf{Outcome}^2_\sigma$ be the expected outcomes obtained by running Algorithm 4.1 on the following page on the corresponding formulae. The serial attack tree computing model is consistent and*

$$\mathsf{Outcome}^1_\sigma = \mathsf{Outcome}^2_\sigma .$$

Proof. Assume that we are processing Algorithm 4.1 on the next page and we have already tried the elementary attacks $X_{\sigma(1)}, \ldots, X_{\sigma(i-1)}$. We see that the node $X_{\sigma(i)}$

Algorithm 4.1: Attacking algorithm in the serial model

```
Input: The set of elementary attacks $S = \{X_1, \dots, X_m\}$,
Input: permutation $\sigma \in \mathfrak{S}_m$ and
Input: a monotone Boolean formula $\mathcal{F}$ describing the attack scenarios.
 1: for $i := 1$ to $m$ do
 2:     Consider $X_{\sigma(i)}$
 3:     if success or failure of $X_{\sigma(i)}$ has no effect on the success or failure of the root node
        then
 4:         Skip $X_{\sigma(i)}$
 5:     else
 6:         Try to perform $X_{\sigma(i)}$
 7:         if the root node succeeds or fails then
 8:             return
 9:         end if
10:     end if
11: end for
```

may be skipped in line 3 if for all the values of $X_{\sigma(i+1)}, \dots, X_{\sigma(n)}$ we have

$$\begin{aligned} &\mathcal{F}\left(X_{\sigma(1)}, \dots, X_{\sigma(i-1)}, \mathsf{t}, X_{\sigma(i+1)}, \dots, X_{\sigma(n)}\right) = \\ = \ &\mathcal{F}\left(X_{\sigma(1)}, \dots, X_{\sigma(i-1)}, \mathsf{f}, X_{\sigma(i+1)}, \dots, X_{\sigma(n)}\right) . \end{aligned}$$

There is no need to proceed after the node $X_{\sigma(i)}$ in line 7, if for all the values of $X_{\sigma(i+1)}, \dots, X_{\sigma(n)}$ we have

$$\begin{aligned} &\mathcal{F}\left(X_{\sigma(1)}, \dots, X_{\sigma(i-1)}, X_{\sigma(i)}, X_{\sigma(i+1)}, \dots, X_{\sigma(n)}\right) = \mathsf{t} \quad \text{or} \\ &\mathcal{F}\left(X_{\sigma(1)}, \dots, X_{\sigma(i-1)}, X_{\sigma(i)}, X_{\sigma(i+1)}, \dots, X_{\sigma(n)}\right) = \mathsf{f} . \end{aligned}$$

We can now see that Algorithm 4.1 really does not depend on the attack tree $\mathcal{T}$ having a particular structure or even $\mathcal{F}$ having any tree structure at all. All the decisions to skip some attack or stop the attacking can be taken just by evaluating the Boolean function $\mathcal{F}$.

By following the similar argument as in Theorem 3.1 on page 46, we can conclude that the serial model also has associativity, commutativity and distributivity properties and the requirements of Mauw and Oostdijk for the attack tree model consistency are also fulfilled. □

Therefore, we can say that this serial attack tree model corresponds to the requirements of Mauw and Oostdijk. Next we will show formally that introducing order to the elementary attacks really increases the attacker's expected outcome compared to the parallel attack tree.

THEOREM 4.2: *Let $\mathcal{F}$ be a monotone Boolean function on variables $X = \{X_1, \ldots, X_n\}$, $n \geq 2$ describing the attack scenarios. Let* $\mathsf{Outcome}_{S,\sigma}$ *be defined by formula (4.1) and let* $\mathsf{Outcome}_S$ *be defined by formula (3.2) for attack suite $S = X$. Then we have*

$$\mathsf{Outcome}_{S,\sigma} \geq \mathsf{Outcome}_S \,. \tag{4.2}$$

If for all the elementary attacks X_i ($i = 1, \ldots, n$) one also has $\mathsf{Expenses}_i > 0$, *then strict inequality holds in formula (4.2).*

Proof. First we note that by formula (3.3) we can compute the success probability of the attacker as follows:

$$p_S = \sum_{\substack{S \subseteq X \\ \mathcal{F}(S := \mathsf{true}) = \mathsf{true}}} \prod_{X_i \in S} p_i \prod_{X_j \in X \setminus S} (1 - p_j) \,,$$

where $\mathcal{F}(S := \mathsf{true})$ denotes the evaluation of the Boolean function $\mathcal{F}$, when all the variables of S are assigned the value true and all others the value false. This is exactly the total probability of all the successful branches of Algorithm 4.1 on the facing page and thus $p_S = p_\sigma$ (implying that p_σ is actually independent of σ). We also have that $\forall\, i\ p_{\sigma,i} \leq 1$ and hence the inequality (4.2) follows.

Assume now that for all X_i we have $\mathsf{Expenses}_i > 0$. Then in order to prove that strict inequality holds in (4.2), we need to show that there exists such an index i that $p_{\sigma,i} < 1$.

Consider the elementary attack $X_{\sigma(n)}$ that the attacker is supposed to try last. If there exists an evaluation of the Boolean variables $X_{\sigma(1)}, \ldots, X_{\sigma(n-1)}$ such that

$$\mathcal{F}\left(X_{\sigma(1)}, \ldots, X_{\sigma(n-1)}, \mathsf{t}\right) = \mathcal{F}\left(X_{\sigma(1)}, \ldots, X_{\sigma(n-1)}, \mathsf{f}\right) \,,$$

then $X_{\sigma(n)}$ is superfluous in this scenario and hence $p_{\sigma,n} < 1$.

If on the other hand we have

$$\mathcal{F}\left(X_{\sigma(1)}, \ldots, X_{\sigma(n-1)}, \mathsf{t}\right) \neq \mathcal{F}\left(X_{\sigma(1)}, \ldots, X_{\sigma(n-1)}, \mathsf{f}\right)$$

for all evaluations of $X_{\sigma(1)}, \ldots, X_{\sigma(n-1)}$, then due to monotonicity of $\mathcal{F}$ we can only have that

$$\mathcal{F}\left(X_{\sigma(1)}, \ldots, X_{\sigma(n-1)}, \mathsf{f}\right) = \mathsf{f}$$

and

$$\mathcal{F}\left(X_{\sigma(1)},\ldots,X_{\sigma(n-1)},\mathsf{t}\right)=\mathsf{t}\,,$$

implying

$$\mathcal{F}\left(a_1,\ldots,a_n\right)\equiv a_n\,.$$

But in this case all the elementary attacks before the last one get skipped, so $p_{\sigma,1} = \ldots = p_{\sigma,n-1} = 0$. $\square$

4.3 Computation Algorithm

There are $n+1$ parameters that need to be computed in order to find the expected outcome using formula (4.1)) – the total success probability p_σ and the probabilities $p_{\sigma,i}$ that the node X_i is encountered during Algorithm 4.1 on page 68. It turns out that there is an efficient algorithm for computing these quantities provided that the given monotone Boolean function can actually be described by a tree. In what follows we will also assume that the tree is binary and that we have the attack suite $S = X$, but this restriction is not a crucial one.

So let us have an attack tree with the set of leaf nodes $X = \{X_1,\ldots,X_n\}$ and the corresponding success probabilities p_i, $i = 1,\ldots,n$. We will assume that all these probabilities are independent and consider the permutation σ of the set X (i.e. $\sigma \in \mathfrak{S}_n$). In order to explain the algorithm, we first introduce three extra parameters to each node Y, namely Y^1, Y^0 and Y^u, showing the probabilities that the node has been proven to be respectively true, false or yet undefined in the course of the analysis.

Initially, we will set $Y^1 = Y^0 = 0$ and $Y^u = 1$ for all the nodes and the algorithm will work by incrementally adjusting these values, so that at the end of the process we will have $R^1 = p_\sigma$ for the root node R. Throughout the computations we will of course retain the invariant $Y^1 + Y^0 + Y^u = 1$ for all the nodes Y, hence one of these parameters is actually superfluous. In the presentation version of the algorithm we will drop the parameter Y^u, even though it actually plays the central role.

Going back to the high-level description of Algorithm 4.1 on page 68, we see that the most difficult decision is in line 3, where the attacker is supposed to find out whether the next elementary attack in his list may have any effect on the success or failure of the root node. An elementary attack does not have any effect iff there is a node on the path from that particular leaf to the root that has already been proven to be true or false. Thus the next elementary attack should be tried iff all of the nodes on this path are undefined – and this is precisely the event that gives us the required probability $p_{\sigma,i}$.

Let the path from the root R to the leaf X_i then be $(Y_0 = R, Y_1, \ldots, Y_m = X_i)$. Thus, we need to compute the probability

$$\begin{aligned}
p_{\sigma,i} &= \Pr[Y_0 = \mathsf{u} \;\&\; Y_1 = \mathsf{u} \;\&\; \ldots \;\&\; Y_m = \mathsf{u}] = \\
&= \Pr[Y_0 = \mathsf{u} \mid Y_1 = \mathsf{u}, \ldots, Y_m = \mathsf{u}] \cdot \\
&\quad \cdot \Pr[Y_1 = \mathsf{u} \mid Y_2 = \mathsf{u}, \ldots, Y_m = \mathsf{u}] \cdot \cdots \\
&\quad \cdots \cdot \Pr[Y_{m-1} = \mathsf{u} \mid Y_m = \mathsf{u}] \cdot \Pr[Y_m = \mathsf{u}] = \\
&= \Pr[Y_0 = \mathsf{u} \mid Y_1 = \mathsf{u}] \cdot \Pr[Y_1 = \mathsf{u} \mid Y_2 = \mathsf{u}] \cdot \cdots \\
&\quad \cdots \cdot \Pr[Y_{m-1} = \mathsf{u} \mid Y_m = \mathsf{u}] \cdot \Pr[Y_m = \mathsf{u}] \,. \qquad (4.3)
\end{aligned}$$

The equations

$$\Pr[Y_k = \mathsf{u} \mid Y_{k+1} = \mathsf{u}, \ldots, Y_m = \mathsf{u}] = \Pr[Y_k = \mathsf{u} \mid Y_{k+1} = \mathsf{u}]$$

hold due to the tree structure of our underlying RDAG and the independence assumption of the elementary attacks.

In formula (4.3) we have $\Pr[Y_m = \mathsf{u}] = \Pr[X_i = \mathsf{u}] = 1$ and all the other probabilities are of the form $\Pr[Y_k = \mathsf{u} \mid Y_{k+1} = \mathsf{u}]$. Hence, we need to evaluate the probability that the parent node Y_k is undefined assuming that one of its children, Y_{k+1}, is undefined. This probability now depends on whether Y_k is an AND- or OR-node. If Y_k is an AND-node and Y_{k+1} is undefined, then so is Y_k if its other child Z is either true or undefined, which is the case with probability $Z^1 + Z^u = 1 - Z^0$. Similarly, if Y_k is an OR-node and Y_{k+1} is undefined, then so is Y_k if its other child Z is either false or undefined, which is the case with the probability $Z^0 + Z^u = 1 - Z^1$.

This way, formula (4.3) gives an efficient way of computing $p_{\sigma,i}$ assuming that the current parameters of the internal nodes of the tree are known. Hence, we need the routines to update these as well. These routines are straightforward. If the elementary attack X_i is tried, only the parameters of the nodes on the path $(Y_m = X_i, \ldots, Y_1, Y_0 = R)$ from that leaf to the root need to be changed. We do it by first setting $Y_m^1 = p_i$, $Y_m^0 = 1 - p_i$ and $Y_m^u = 0$ and then proceed towards the root. If the node we encounter is AND-node A with children B and C, we set

$$A^1 = B^1 \cdot C^1 \,, \qquad (4.4)$$
$$A^0 = B^0 + C^0 - B^0 \cdot C^0 \,, \qquad (4.5)$$

Algorithm 4.2: Computing the probabilities $p_{\sigma,i}$

```
Input: An attack tree with leaf set $X = \{X_1, \dots, X_n\}$
Input: Permutation $\sigma \in \mathfrak{S}_n$
Output: The probabilities $p_{\sigma,i}$ for $i = 1, \dots, n$
 1: for all $X_i \in \{X_1, \dots, X_n\}$ do
 2:     $X_i^1 := 0$ and $X_i^0 := 0$
 3: end for
 4: for $i := 1$ to $n$ do
 5:     Find the path $(Y_0, \dots, Y_m)$ from the root $Y_0 = R$ to the leaf $Y_m = X_{\sigma(i)}$
 6:     $X^1_{\sigma(i)} := p_{\sigma(i)}$
 7:     $X^0_{\sigma(i)} := 1 - p_{\sigma(i)}$
 8:     Compute
            $p_{\sigma,\sigma(i)} := \prod_{j=1}^{m} (1 - Z_j^a),$
        where $Z_j$ is the sibling node of $Y_j$ and
            $a = \begin{cases} 1 & \text{if } Y_{j-1} \text{ is an OR-node} \\ 0 & \text{if } Y_{j-1} \text{ is an AND-node} \end{cases}$
 9:     Update the parameters of the nodes $Y_{m-1}, Y_{m-2}, \dots, Y_0$ according to formulae (4.4–4.7)
10: end for
```

and if we encounter an OR-node A with children B and C, we set

$$A^1 = B^1 + C^1 - B^1 \cdot C^1 , \qquad (4.6)$$

$$A^0 = B^0 \cdot C^0 . \qquad (4.7)$$

As noted above, we see that the quantities Y^u are actually never needed in the computations.

In order to compute the $n+1$ necessary probabilities Algorithm 4.2 makes one run through all the leaves of the tree and at each run the path from the leaf to the root is traversed twice. Since the number of vertices on such a path in a (binary) tree can not be larger than the number of leaves n, we get that the worst-case time complexity of Algorithm 4.2 is $\mathcal{O}(n^2)$. If the tree is roughly balanced, this estimate drops even to $\mathcal{O}(n \log n)$. This is a huge performance increase compared to a naïve algorithm that one could design based on the complete attack scenario.

4.4 Outcome Results Comparison

Theorem 4.2 on page 69 states that the serial model yields greater outcome values when all elementary attack expenses are greater than zero. In order to get an idea of how much the serial model actually gains when compared to the parallel one, we tested their outcomes on random trees. However, to meaningfully compare the outcome values, there are some details to be discussed.

It may occur that the expected outcome computed by the parallel routine may be negative (or even exactly zero), but the outcome in the serial model may still be positive. In such a situation, it is hard to quantify the percentage of actual improvement of the result. It is possible to measure the absolute increase in the outcome, but in this case, it is hard to relate improvements on the different trees.

Therefore, we first picked trees with positive outcome values computed by the parallel model and then normalized the tree input parameters, so that the outcome value was exactly 1000. If the original outcome was k, then we simply multiplied all monetary parameters by $1000/k$. We then computed the outcome value for the serial model.

Altogether, for every $n \in \{5, 6, \dots, 13\}$ we generated 53 normalized trees with n leaves and computed the average outcome with the serial model. The results are displayed in Figure 4.2 on the following page. However, it seems that there is no clear trend in the data. The increase varies greatly and the standard deviation exceeds the mean value in many datapoints.

In many cases, when comparing the serial model outcome to the parallel model outcome, there was actually no improvement at all. This can be the case when the optimal attack strategy for both the parallel and serial models consists only of one elementary attack. For relatively small random trees, this happens quite often, however, in practical attack scenarios, it is relatively unlikely that the system can be attacked with just a single elementary attack without any dependencies. Therefore, it could be the case that this kind of testing with the random trees does not correspond well to reality.

4.5 Genetic Algorithm

It is clear from previous sections that exact algorithms do not work on the serial attack model quickly, in fact, they are rather slow and only capable of processing attack trees up to 13 leaves in the reasonable time. Therefore, we will once again

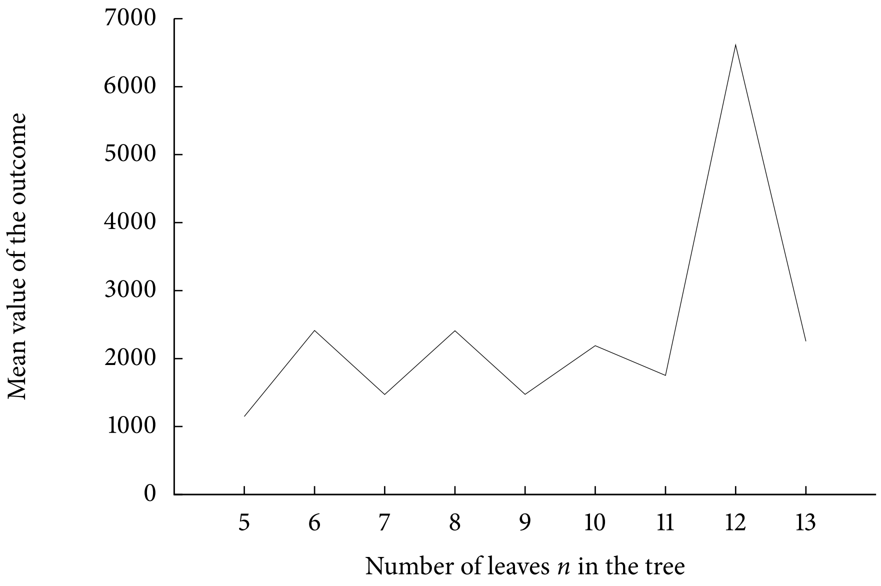


FIGURE 4.2: Mean expected outcome of the serial model on normalized trees where the parallel model gives the outcome value 1000.

turn to approximations and will use the similar genetic algorithm method from Section 3.7 on page 58.

With the current model, however, we also need to evolve the order of the attacks. We refer to the overview in [49] by Larrañaga *et al.*, where solving the Traveling Salesman Problem (TSP) with the genetic algorithm is being studied. Altogether 25 ways are presented to encode the solution for the genetic algorithm and to define crossover and mutate operator, which gives us plenty of choices and examples. The problem is that in the classical TSP problem, the parent permutations are always defined on the same set, because the traveling salesman has to visit all cities only once.

In our case, parent attack suites may have a different number of attacks and therefore, the permutations also might be of different length. Additionally, it does not make sense to evolve the subsets S and permutations σ separately, because then we would need to run the genetic algorithm separately for each subset of X and then choose the subset with the best fitness function. We need to evolve both the elements of the subsets and the order of those elements at the same time for

Algorithm 4.3: Creating random permutation σ of the set $S = \{X_1, \ldots, X_m\}$ (from [45, pp. 145–146])

```
Input: The size of set $S = m$
Output: $\sigma$ is the uniformly random permutation of sequence $\langle 1, \ldots, m \rangle$
1: Procedure RandomPermutation($m$)
2: $\sigma(1) := 1$
3: for $i := 2$ to $m$ do
4:     $j := \mathsf{random}(1 \ldots i)$
5:     $\sigma(i) := \sigma(j)$
6:     $\sigma(j) := j$
7: end for
8: return $\sigma$
```

our genetic algorithm approach to be useful.

To encode the ordered attack suite (S, σ) for the genetic algorithm we will use the same bit array approach as with the parallel model. The permutation σ will be represented by the array of numbers.

4.5.1 Generating Initial Population

To start the genetic algorithm, we need an initial population of ordered attack suites. In the parallel model, we created satisfying attack suites by recursively traversing the tree $\mathcal{T}$ and in and-nodes chose all the children and in or-nodes chose randomly at least one of the children. The resulting attack suite was guaranteed to be a "live" individual.

For the serial model, we need to create the S and also the $\preccurlyeq_S$. We will use the same Algorithm 3.4 on page 60 for creating the S, and then we will generate a random permutation σ. We will use the Fisher-Yates shuffle algorithm from [45, pp. 145–146] described in Algorithm 4.3 for ensuring that each combination has the same probability to appear.

4.5.2 Crossing and mutating individuals

We have two ordered attack suites (S_1, σ_1) and (S_2, σ_2), which will have a descendant (S, σ).

To create the set S, we will use the same technique as in Section 3.7 on page 58. We will go over all the n elementary attacks in the set X and will decide randomly from which parent we will take the status of the elementary attack X_i. For example, if we decide to use parent S_1 for the i and $X_i \in S_1$, then $X_i \in S$ as well. Therefore those elementary attacks, which are included in both parents, will be included in the descendant for sure. Those attacks which are included in one of the parents will have 0.5 chance of being included in the descendant and those attacks which are in neither parents will have 0 chance of being included.

Because we are not able to use the natural crossover operation on the ordered attack suites, we will simply randomly pick permutation elements from parent permutations and look out for duplicate elements. This is based on the alternating position crossover operator defined in [49, pp. 23] and we modify it to work for our case.

We present both procedures in Algorithm 4.4 on the next page. To illustrate the workings of the algorithm, let us consider the following example.

Let us take $X = \{X_1, \dots, X_5\}$ and for two parents

$$\begin{aligned} S_1 &= \{X_1, X_2, X_4, X_5\} & \sigma_1 &= \langle 1, 2, 4, 5 \rangle \, , \\ S_2 &= \{X_3, X_4\} & \sigma_2 &= \langle 4, 3 \rangle \, . \end{aligned}$$

Let us assume that when composing the descendant S from elementary attacks, we randomly chose the corresponding parents and we got the following resultant set:

$$S = \{X_1, X_3, X_4, X_5\} \, .$$

Let us also assume that we did not mutate the set anymore. Then the permutation σ of the descendant will be randomly chosen from parent permutations:

1. We randomly chose σ_2 and got element 4 from there. $\sigma = \langle 4 \rangle$ and $\sigma_2 = \langle 3 \rangle$.
2. We randomly chose σ_1 and got element 1 from there. $\sigma = \langle 4, 1 \rangle$ and $\sigma_1 = \langle 2, 4, 5 \rangle$.
3. Next, we chose again σ_1 and got element 2 from there, however $X_2 \notin S$ and therefore we just skip it. $\sigma = \langle 4, 1 \rangle$ and $\sigma_1 = \langle 4, 5 \rangle$.
4. Again, we chose σ_1 and the next element is 4, but σ already contains 4. Therefore we just throw it away. $\sigma = \langle 4, 1 \rangle$ and $\sigma_1 = \langle 5 \rangle$.
5. Then, σ_1 is chosen again, and the last element 5 is added to σ. We now have $\sigma = \langle 4, 1, 5 \rangle$ and $\sigma_1 = \langle \rangle$.
6. Because σ_1 is now empty, we will just get the remaining elements from σ_2

ALGORITHM 4.4: Creating descendant attack suite from two parent attack suites

```
INPUT:  The set $X$ of all elementary attacks
INPUT:  Ordered attack suites $(S_1, \sigma_1)$ and $(S_2, \sigma_2)$
OUTPUT:  Descendant ordered attack suite $(S, \sigma)$
  1: PROCEDURE CrossSuites$(X, S_1, \sigma_1, S_2, \sigma_2)$
  2: FOR $i := 1$ TO $n$ DO
  3:     $k :=$ random$(1, 2)$
  4:     IF $X_i \in S_k$ THEN
  5:         add $X_i$ to $S$
  6:     END IF
  7:     IF we have decided to mutate the $S$ AND decide to mutate the bit $i$ THEN
  8:         flip the status of the $X_i \in S$
  9:     END IF
 10: END FOR
 11: make copies $\beta_1$ and $\beta_2$ from permutations $\sigma_1$ and $\sigma_2$
 12: $i := 1$
 13: WHILE $\beta_1 \neq \varnothing$ AND $\beta_2 \neq \varnothing$ DO
 14:     $k :=$ random$(1, 2)$
 15:     IF $\beta_k = \varnothing$ THEN
 16:         choose another parent for $k$
 17:     END IF
 18:     IF $\beta_k(1) \in S$ AND $\beta_k(1) \notin \sigma$ THEN
 19:         $\sigma(i) := \beta_k(1)$
 20:         $i := i + 1$
 21:     END IF
 22:     remove $\beta_k(1)$ from $\beta_k$
 23: END WHILE
 24: RETURN  $S$ and $\sigma$
```

and because they are not already in σ, we add them to σ.

7. Finally, $\sigma = \langle 4, 1, 5, 3 \rangle$.

To choose the best individuals after the crossover, we simply compute Outcome$_{S,\sigma}$ for each ordered attack suite $\langle S, \sigma \rangle$ and sort out the h best attack suites for the next generation.

4.5.3 Analysis

We now know how to produce the initial generation with Algorithm 4.3 on page 75 and we know how to cross the ordered attack suites with each other with Algorithm 4.4 on the previous page. For the fitness function, we will just compute $\mathsf{Outcome}_{S,\sigma}$ and then sort out the h best attack suites. The procedure is listed as Algorithm 4.5 on the facing page.

The complexity of this genetic algorithm in the worst case is $\mathcal{O}(g(nh^2+n^2h^2-\frac{n^2h}{2}+h^2\log h))$ because we will generate g generations, we will spend $\mathcal{O}(nh^2)$ in the crossover section, after that we need to compute $\mathsf{Outcome}_{S,\sigma}$ for $h^2-\frac{h}{2}$ new individuals, each of them takes $\mathcal{O}(n^2)$ steps in the worst case and finally we need to sort out h best individuals, which takes $\mathcal{O}(h^2\log h)$ steps.

To verify that the genetic algorithm works in practice and finds the correct attack suite, we used the same database of 6000 attack trees and computed the exact result for the tree and then ran genetic algorithm with various parameters on the same tree and compared the results. Because of the performance of the serial attack tree algorithm, we were only able to test on the trees having up to 12 leaves. The results for the parameter functions $h=2n$ and $g=2n$ and $h=n$ and $g=n$ are given in Figure 4.4 on page 81.

It turns out that we are able to get almost the same level of accuracy from the genetic algorithm as we did with the parallel attack tree models. The complexity of the genetic algorithm in the case of $h=2n$ and $g=2n$ is $\mathcal{O}(n^5)$. The performance results are given in Figure 4.3 on page 80.

Algorithm 4.5: Finding approximate best ordered attack suite S and permutation σ for the attack tree $\mathcal{T}$ using the genetic algorithm

```
Input: and-or-tree $\mathcal{T}$ with estimated parameters,
Input: Size of the population as $h$ and number of generations as $g$
Output: $\mathsf{Outcome}_{S,\sigma}$ is approximately the best outcome for attack tree $\mathcal{T}$
1: Procedure ComputeApproximateOrderedAttackSuite()
2: Generate $h$ individual ordered attack suites by calling Algorithm 4.3 on page 75
3: for $i = 2$ to $g$ do
4:     Cross and mutate $h$ attack suites with each other by calling Algorithm 4.4 on
       page 77, producing $\binom{h}{2}$ new attack suites
5:     Add $h$ attack suites from the generation $g_{i-1}$ to the current generation $g_i$
6:     Filter out those individuals, who are live (i.e. $\mathcal{F}(S_j := \mathtt{t}) = \mathtt{t}$))
7:     Compute the $\mathsf{Outcome}_{S_j,\sigma_j}$ for all individuals and choose $h$ best individuals for
       the next generation $g_{i+i}$
8: end for
9: return Best ordered attack suite $\langle S, \sigma \rangle$ having the largest $\mathsf{Outcome}_{S,\sigma}$ from the last
   generation $g_g$
```

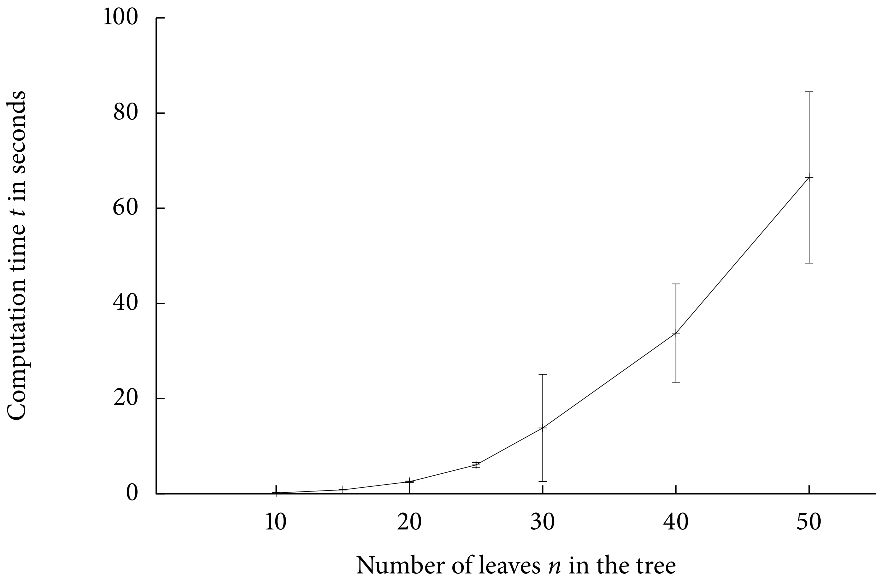


FIGURE 4.3: Performance results of Algorithm 4.5 on the previous page in case of parameters $h = 2n$ and $g = 2n$. The confidence interval for the attack tree computation time t is given with 95% confidence level. Performance tests were done on 2.33 GHz Pentium Xeon machine.

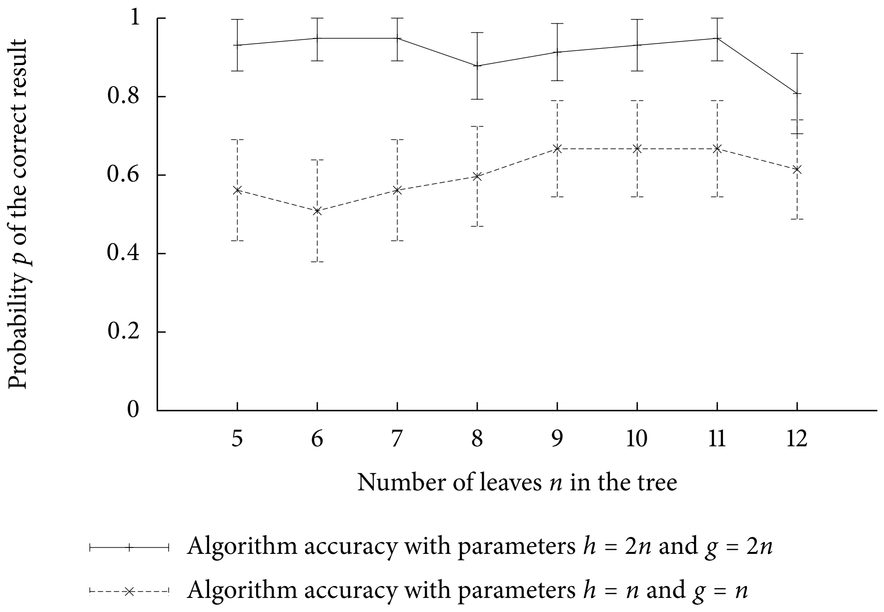


FIGURE 4.4: Accuracy estimation of Algorithm 4.5 on page 79 for different values of population size and number of generations. The confidence interval for probability p is given with the confidence level 95%.

Conclusions and Future Research

This thesis studied two multi-parameter attack tree models. The first model considered all elementary attacks occurring at the same time, in parallel, and the second allowed the attacker to choose the order of the attacks and then try them sequentially.

Both models proved to be consistent by Theorem 3.1 on page 46 and Theorem 4.1 on page 67, in terms of requirements of Mauw and Oostdijk. This means that the outcome values of two equivalent attack trees which describe the same attack are equal and it also confirms that the outcome value of the attack tree does not depend on the particular structure or order of attacks in the tree. The attack tree can be represented by the corresponding Boolean formula and usual transformations can be applied to that Boolean formula and the resulting outcome value will be unchanged.

It was also shown by Theorem 3.2 on page 47 and Theorem 4.2 on page 69 that both models result at least as great outcome values, as the previous multi-parameter attack tree model by Buldas *et al.*. This means that in our models the attacker is able make as good and sometimes better decisions regarding which attacks to choose as in the previous multi-parameter attack tree model. However, because our computing model is not using the parameter propagation from the attack tree leaves up to the root node, we are able to find a global maximum over all possible solutions and do not have to do local optimum based decisions in OR-nodes. In certain cases, choosing more than one attack from OR-node allows the attacker to increase the success probability of the chosen attack suite and in this way increase the expected outcome value.

With the serial model, we allow the attacker to use even more information for making decisions. As shown in Theorem 4.2 on page 69, the serial model will yield strictly greater expected outcome than the parallel model if all the elementary attacks have non-zero costs. Because we are only considering semi-adaptive attackers in the serial model, the models with fully adaptive attackers could result in even greater expected outcome values.

Unfortunately, considering all possible solutions and then choosing the global maximum over those means that the complexity of the outcome computation by Algorithm 3.2 on page 54 and Algorithm 4.1 on page 68 is exponential to the number of elementary attacks in the attack tree and this limits the application of those models in the real life situations, since it would take simply too long to compute the exact outcome value. It would be useful if we could give at least the approximate outcome value of the attack tree in a few minutes. The genetic Algorithm 3.5 on page 61 and Algorithm 4.5 on page 79 were developed and they proved to be very successful with our testing data-set, giving the same outcome result as exact algorithms in about 90% of cases. The running time for the genetic algorithm is about 5 minutes for attack trees with 100 leaves, which allows the attack tree analysis to be included in the real life security analysis.

Attack tree analysis and quantitative risk assessment is still a relatively young field and much remains to be discovered. We agree with Verendel who argues in [78] that quantified security models have not yet been corroborated by predicting outcomes of experiments in repeated large-sample tests. Real-life tests and the evaluation of required attack tree parameters is still a weak point in our models as well and there is certainly more research to be done. To find out how to successfully evaluate those parameters, collaboration with experts from different fields, such as micro-economy, psychology, law enforcement, security analysis, and system design and administration, *etc.*, is essential.

Even after successfully estimating some of the attack tree parameters, those estimations, and the computation results based on those estimations, should not be considered exact point-values. When the estimation error and the confidence level of the human expert could be taken into account, the attack tree utility value is also not the exact point-value. Naturally, decision makers would like to know the confidence limits of the results, but the sensitivity of the attack tree computations to the parameter errors has not yet been studied. This remains an important requirement for the application of the attack tree analysis in the real world.

Another interesting research problem is to combine the attack tree analysis with choosing the cheapest set of security measures for systems. This would allow managers to make more vested investment decisions.

Even with those not yet answered questions we suggest that the attack tree models presented in this thesis are useful additions to the attack tree analysis.

Index

References

[1] A. Agresti and B. Coull. Approximate is better than "exact" for interval estimation of binomial proportions. *The American Statistician*, 52(2):119–126, 1998.

[2] A. Aijaz, B. Bochow, F. Dötzer, A. Festag, M. Gerlach, R. Kroh, and T. Leinmüller. Attacks on inter vehicle communication systems – an analysis. In *Proceedings of the 3rd International Workshop on Intelligent Transportation*, pages 189–194, 2006.

[3] P. Ammann, D. Wijesekera, and S. Kaushik. Scalable, graph-based network vulnerability analysis. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 217–224. ACM Press, 2002.

[4] A. Andrusenko, A. Jürgenson, and J. Willemson. Serial model for attack tree computations. *KSII Transactions on Internet and Information Systems*, 2010. (*to appear*).

[5] E. Baadshaug, G. Erdogan, and P. Meland. Security modeling and tool support advantages. In *ARES '10: International Conference on Availability, Reliability, and Security*, pages 537–542, 2010.

[6] G. S. Becker. Crime and punishment: An economic approach. *Journal of Political Economy*, 76:169, 1968.

[7] S. Bistarelli, M. Dall'Aglio, and P. Peretti. Strategic games on defense trees. In T. Dimitrakos, F. Martinelli, P. Y. A. Ryan, and S. A. Schneider, editors, *Formal Aspects in Security and Trust (FAST 2006)*, volume 4691 of LNCS, pages 1–15. Springer-Verlag, 2006.

[8] R. Browne. C4I defensive infrastructure for survivability against multi-mode attacks. In *Proceedings of 21st Century Military Communications Conference (MILCOM 2000)*, volume 1, pages 417–424, 2000.

[9] A. Buldas and A. Jürgenson. Does secure time-stamping imply collision-free hash functions? In W. Susilo, J. K. Liu, and Y. Mu, editors, *Proceedings of International Conference on Provable Security (ProvSec 2007)*, volume 4784 of LNCS, pages 138–150. Springer-Verlag, 2007.

[10] A. Buldas and T. Mägi. Practical security analysis of e-voting systems. In A. Miyaji, H. Kikuchi, and K. Rannenberg, editors, *Advances in Information and Computer Security: Second International Workshop on Security (IWSEC 2007)*, volume 4752 of LNCS, pages 320–335. Springer-Verlag, 2007.

[11] A. Buldas, P. Laud, J. Priisalu, M. Saarepera, and J. Willemson. Rational choice of security measures via multi-parameter attack trees. In *First International Workshop on Critical Information Infrastructures Security (CRITIS 2006)*, volume 4347 of LNCS, pages 235–248. Springer-Verlag, 2006.

[12] A. Buldas, A. Jürgenson, and M. Niitsoo. Efficiency bounds for adversary constructions in black-box reductions. In *ACISP '09: Proceedings of the 14th Australasian Conference on Information Security and Privacy*, pages 264–275. Springer-Verlag, 2009.

[13] E. Byres, M. Franz, and D. Miller. The use of attack trees in assessing vulnerabilities in SCADA systems. In *International Infrastructure Survivability Workshop (IISW'04), IEEE*, 2004.

[14] J. Collet. Some remarks on rare-event approximation. *IEEE Transactions on Reliability*, 45(1):106–108, 1996.

[15] S. Convery, D. Cook, and M. Franz. An attack tree for the Border Gateway Protocol, 2004. Available at `http://www.ietf.org/proceedings/04aug/I-D/draft-ietf-rpsec-bgpattack-00.txt`.

[16] M. Dacier. *Towards Quantitative Evaluation of Computer Security*. PhD thesis, Institut National Polytechnique de Toulouse, 1994.

[17] K. Daley, R. Larson, and J. Dawkins. A structural framework for modeling multi-stage network attacks. In *Proceedings of International Conference on Parallel Processing Workshops*, pages 5–10, 2002.

[18] G. C. Dalton, K. S. Edge, R. F. Mills, and R. A. Raines. Analysing security risks in computer and Radio Frequency Identification (RFID) networks using attack and protection trees. *International Journal of Security and Networks*, 5(2/3):87–95, 2010.

[19] R. Dantu, K. Loper, and P. Kolan. Risk management using behavior based attack graphs. In *ITCC '04: Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'04)*, volume 1, page 445. IEEE Computer Society, 2004.

[20] L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold Company, first edition, 1991.

[21] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, 1962.

[22] M. H. Diallo, J. Romero-mariona, S. E. Sim, T. A. Alspaugh, and D. J. Richardson. A comparative evaluation of three approaches to specifying security requirements. In *Proceedings of the Twelfth Working Conference on Requirements Engineering: Foundation for Software Quality*, 2006.

[23] K. Edge, G. Dalton, R. Raines, and R. Mills. Using attack and protection trees to analyze threats and defenses to homeland security. In *Military Communications Conference (MILCOM 2006)*, pages 1–7. IEEE Computer Society, 2006.

[24] K. Edge, R. Raines, R. Baldwin, and M. Grimaila. Analyzing security measures for mobile ad hoc networks using attack and protection trees. In *2nd International Conference on i-Warfare and Security*, pages 47–56, 2007.

[25] K. Edge, R. Raines, M. Grimaila, R. Baldwin, R. Bennington, and C. Reuter. The use of attack and protection trees to analyze security for an online banking system. In *Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS 2007)*, page 144b. IEEE Computer Society, 2007.

[26] I. N. Fovino and M. Masera. Through the description of attacks: A multidimensional view. In *Computer Safety, Reliability, and Security*, volume 4166 of LNCS, pages 15–28. Springer-Verlag, 2006.

[27] I. N. Fovino and M. Masera. Parameters for quantitative security assessment of complex systems. In *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE Computer Society, 2007.

[28] I. N. Fovino, M. Masera, and A. D. Cian. Integrating cyber attacks within fault trees. *Reliability Engineering and System Safety*, 94(9):1394–1402, 2009.

[29] M. Frigault, L. Wang, A. Singhal, and S. Jajodia. Measuring network security using Dynamic Bayesian Network. In *QoP '08: Proceedings of the 4th ACM workshop on Quality of Protection*, pages 23–30. ACM, 2008.

[30] R. Frigg and S. Hartmann. Models in science. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Summer 2009 edition, 2009.

[31] C. Fung, Y.-L. Chen, X. Wang, J. Lee, R. Tarquini, M. Anderson, and R. Linger. Survivability analysis of distributed systems using attack tree methodology. In *Military Communications Conference (MILCOM 2005)*, volume 1, pages 583–589. IEEE Computer Society, 2005.

[32] V. Gandotra, A. Singhal, and P. Bedi. Identifying security requirements hybrid technique. In *ICSEA '09: Proceedings of the Fourth International Conference on Software Engineering Advances*, pages 407–412. IEEE Computer Society, 2009.

[33] M. Higuero, J. Unzilla, P. Saiz, E. Jacob, M. Aguado, and I. Goirizelaia. A practical tool for analysis of security in systems for distribution of digital contents based on "attack trees". In *IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB '09)*, pages 1–6, 2009.

[34] M. V. Higuero, J. Unzilla, E. Jacob, P. Sáiz, and D. Luengo. Application of "attack trees" technique to copyright protection protocols using watermarking and definition of a new transactions protocol SecDP (Secure Distribution Protocol). In V. Roca and F. Rousseau, editors, *Multimedia Interactive Protocols and Systems*, volume 3311 of LNCS, pages 264–275. Springer-Verlag, 2004.

[35] K. Ingols, M. Chu, R. Lippmann, S. Webster, and S. Boyer. Modeling modern network attacks and countermeasures using attack graphs. In *ACSAC '09:*

Proceedings of the 2009 Annual Computer Security Applications Conference, pages 117–126. IEEE Computer Society, 2009.

[36] IEC-61025. *Fault Tree Analysis, edition 2.0*. International Electrotechnical Commission, 2006.

[37] K.-W. L. Jeannette and J. Wing. Game strategies in network security. In *Proceedings of the Workshop on Foundations of Computer Security*, pages 1–2, 2002.

[38] A. Jürgenson and J. Willemson. Processing multi-parameter attacktrees with estimated parameter values. In A. Miyaji, H. Kikuchi, and K. Rannenberg, editors, *Advances in Information and Computer Security: Second International Workshop on Security (IWSEC 2007)*, volume 4752 of LNCS, pages 308–319. Springer-Verlag, 2007.

[39] A. Jürgenson and J. Willemson. Computing exact outcomes of multi-parameter attack trees. In R. Meersman and Z. Tari, editors, *On the Move to Meaningful Internet Systems (OTM 2008)*, volume 5332 of LNCS, pages 1036–1051. Springer-Verlag, 2008.

[40] A. Jürgenson and J. Willemson. Ründepuud: pooladaptiivne mudel ja ligikaudsed arvutused (in Estonian). Technical Report T-4-4, Cybernetica, Institute of Information Security, 2009.

[41] A. Jürgenson and J. Willemson. Serial model for attack tree computations. In D. Lee and S. Hong, editors, *Revised Papers from 12th International Conference on Information Security and Cryptology (ICISC 2009)*, volume 5984 of LNCS, pages 118–128. Springer-Verlag, 2010.

[42] A. Jürgenson and J. Willemson. On fast and approximate attack tree computations. In J. Kwak, R. H. Deng, Y. Won, and G. Wang, editors, *Information Security Practice and Experience, 6th International Conference (ISPEC 2010)*, volume 6047 of LNCS, pages 56–66. Springer-Verlag, 2010.

[43] P. Khand. System level security modeling using attack trees. In *2nd International Conference on Computer, Control and Communication (IC4 2009)*, pages 1–6, 2009.

[44] P. A. Khand. Attack tree based cyber security analysis of nuclear digital instrumentation and control systems. *The Nucleus*, 46(4):415–428, 2009.

[45] D. E. Knuth. *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*. Addison-Wesley, third edition, 1997.

[46] I. Kotenko and M. Stepashkin. Attack graph based evaluation of network security. In *Communications and Multimedia Security*, volume 4237 of LNCS, pages 216–227. Springer-Verlag, 2006.

[47] D. Kozen. *The design and analysis of algorithms*. Springer-Verlag, 1992.

[48] K. Kutzkov. New upper bound for the #3-SAT problem. *Information Processing Letters*, 105(1):1–5, 2007.

[49] P. Larrañaga, C. M. H. Kuijpers, R. Murga, I. Inza, and S. Dizdarevic. Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial Intelligence Review*, 13:129–170, 1999.

[50] T. T. C. Lawrence D. Brown and A. DasGupta. Interval estimation for a binomial proportion. *Statistical Science*, 16(2):101–133, 2001.

[51] X. Lin, P. Zavarsky, R. Ruhl, and D. Lindskog. Threat modeling for CSRF attacks. In *CSE '09: Proceedings of the 2009 International Conference on Computational Science and Engineering*, pages 486–491. IEEE Computer Society, 2009.

[52] Y. Luo, F. Szidarovszky, Y. Al-Nashif, and S. Hariri. A game theory based risk and impact analysis method for intrusion defense systems. In *ACS/IEEE International Conference on Computer Systems and Applications*, pages 975–982. IEEE Computer Society, 2009.

[53] S. Mauw and M. Oostdijk. Foundations of attack trees. In D. Won and S. Kim, editors, *International Conference on Information Security and Cryptology (ICISC 2005)*, volume 3935 of LNCS, pages 186–198. Springer-Verlag, 2005.

[54] A. Miede, N. Nedyalkov, C. Gottron, A. Konig, N. Repp, and R. Steinmetz. A generic metamodel for IT security attack modeling for distributed systems. In *ARES '10: International Conference on Availability, Reliability, and Security*, pages 430–437, 2010.

[55] A. Morais, E. Martins, A. Cavalli, and W. Jimenez. Security protocol testing using attack trees. In *Proceedings of the 2009 International Conference on Computational Science and Engineering*, volume 2, pages 690–697. IEEE Computer Society, 2009.

[56] D. Nicol, W. Sanders, and K. Trivedi. Model-based evaluation: from dependability to security. *IEEE Transactions on Dependable and Secure Computing*, 1(1):48–65, 2004.

[57] A. L. Opdahl and G. Sindre. Experimental comparison of attack trees and misuse cases for security threat identification. *Information and Software Technology*, 51(5):916–932, 2009.

[58] J. Pamula, S. Jajodia, P. Ammann, and V. Swarup. A weakest-adversary security metric for network configuration security analysis. In *QoP '06: Proceedings of the 2nd ACM workshop on Quality of protection*, pages 31–38. ACM, 2006.

[59] C. Phillips and L. P. Swiler. A graph-based system for network-vulnerability analysis. In *NSPW '98: Proceedings of the 1998 workshop on New security paradigms*, pages 71–79. ACM, 1998.

[60] P. Ralston, J. Graham, and J. Hieb. Cyber security risk assessment for SCADA and DCS networks. *ISA Transactions*, 46(4):583–594, 2007.

[61] I. Ray and N. Poolsapassit. Using attack trees to identify malicious attacks from authorized insiders. In S. D. C. di Vimercati, P. F. Syverson, and D. Gollmann, editors, *10th European Symposium on Research in Computer Security*, volume 3679 of LNCS, pages 231–246. Springer-Verlag, 2005.

[62] K. Reddy, H. S. Venter, M. Olivier, and I. Currie. Towards privacy taxonomy-based attack tree analysis for the protection of consumer information privacy. In *Proceedings of the 2008 Sixth Annual Conference on Privacy, Security and Trust*, pages 56–64. IEEE Computer Society, 2008.

[63] V. Saini, Q. Duan, and V. Paruchuri. Threat modeling using attack trees. *Journal of Computing Sciences in Colleges*, 23(4):124–131, 2008.

[64] K. Sallhammar. *Stochastic models for combined security and dependability evaluation*. PhD thesis, Norwegian University of Science and Technology, 2007.

[65] C. Salter, O. S. Saydjari, B. Schneier, and J. Wallner. Toward a secure system engineering methodology. In *NSPW '98: Proceedings of the 1998 Workshop on New Security Paradigms*, pages 2–10. ACM, 1998.

[66] S. E. Schechter and M. D. Smith. How much security is enough to stop a thief? the economics of outsider theft via computer systems and networks. In *in Financial Cryptography*, pages 122–137. Springer-Verlag, 2003.

[67] B. Schneier. Attack trees: Modeling security threats. *Dr. Dobb's Journal*, 24 (12):21–29, 1999.

[68] B. Schneier. *Secrets & Lies. Digital Security in a Networked World.* John Wiley & Sons, 2000.

[69] M. Schumacher. *Security Engineering with Patterns: Origins, Theoretical Models, and New Applications*, volume 2754 of LNCS. Springer-Verlag, 2003.

[70] M. Schumacher and U. Roedig. Security engineering with patterns. In *PLoP 2001: 8th Conference on Pattern Languages of Programs*, LNCS. Springer-Verlag, 2001.

[71] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing. Automated generation and analysis of attack graphs. In *SP '02: Proceedings of the 2002 IEEE Symposium on Security and Privacy*, page 273. IEEE Computer Society, 2002.

[72] J. Steffan and M. Schumacher. Collaborative attack modeling. In *SAC '02: Proceedings of the 2002 ACM symposium on Applied Computing*, pages 253–259. ACM, 2002.

[73] C.-W. Ten, C.-C. Liu, and M. Govindarasu. Vulnerability assessment of cybersecurity for SCADA systems using attack trees. In *Power Engineering Society General Meeting, 2007. IEEE*, pages 1–8, June 2007.

[74] T. Tidwell, R. Larson, K. Fitch, and J. Hale. Modeling internet attacks. In *Proceedings of the IEEE Workshop on Information Assurance and Security*, pages 54–59, 2001.

[75] K. Trivedi, D. S. Kim, A. Roy, and D. Medhi. Dependability and security models. In *7th International Workshop on Design of Reliable Communication Networks (DRCN 2009)*, pages 11–20, 2009.

[76] I. Tøndel, J. Jensen, and L. Rstad. Combining misuse cases with attack trees and security activity models. In *ARES '10: International Conference on Availability, Reliability, and Security*, pages 438–445, 2010.

[77] MIL-STD-1785. *System Security Engineering Program Management Requirements.* U.S. Department of Defense, 1989.

[78] V. Verendel. Quantified security is a weak hypothesis: a critical survey of results and assumptions. In *NSPW '09: Proceedings of the 2009 workshop on New security paradigms workshop*, pages 37–50. ACM, 2009.

[79] W. Veseley, F. Goldberg, N. Roberts, and D. Haasl. Fault tree handbook. Technical Report NUREG-0492, U.S. Nuclear Regulatory Commission, 1981.

[80] W. Veseley, M. Stamatelatos, J. Dugan, J. Fragola, J. Minarick, and J. Railsback. *Fault Tree Handbook with Aerospace Applications.* NASA, Office of Safety and Mission Assurance, 2002.

[81] M. Wachter and R. Haenni. Propositional DAGs: A new graph-based language for representing Boolean functions. In P. Doherty, J. Mylopoulos, and C. A. Welty, editors, *Proceedings of Tenth International Conference on Principles of Knowledge Representation and Reasoning*, pages 277–285. AAAI Press, 2006.

[82] A. Wald and J. Wolfowitz. Confidence limits for continuous distribution functions. *The Annals of Mathematical Statistics*, 10(2):105–118, 1939.

[83] L. Wang, S. Noel, and S. Jajodia. Minimum-cost network hardening using attack graphs. *Computer Communications*, 29(18):3812–3824, 2006.

[84] L. Wang, C. Yao, A. Singhal, and S. Jajodia. Interactive analysis of attack graphs using relational queries. In E. Damiani and P. Liu, editors, *20th Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, volume 4127 of LNCS, pages 119–132. Springer-Verlag, 2006.

[85] L. Wang, T. Islam, T. Long, A. Singhal, and S. Jajodia. An attack graph-based probabilistic security metric. In *Proceeedings of the 22nd annual IFIP WG 11.3 working conference on Data and Applications Security*, pages 283–296. Springer-Verlag, 2008.

[86] L. Wang, C. Yao, A. Singhal, and S. Jajodia. Implementing interactive analysis of attack graphs using relational databases. *Journal of Computer Security*, 16(4):419–437, 2008.

[87] J. D. Weiss. A system security engineering process. In *Proceedings of the 14th National Computer Security Conference*, pages 572–581, 1991.

[88] R. R. Yager. OWA trees and their role in security modeling using attack trees. *Information Sciences*, 176(20):2933–2959, 2006.

Abstract

Measuring information security is hard. This is because it is hard to estimate how likely it is for the complicated, multi-stage attacks to succeed and how much extra security those deployed measures really give you. Because you do not have ages of historical data and details about incidents are regularly not shared, you cannot use frequentist approach to predict the annual expected loss. Engineers have solved the similar problem of estimating the reliability of an individual system with fault trees and the same principles have been applied to information security as well under the name of attack trees. In the attack tree analysis you specify the attacker's goal as the root of the tree and then progressively try to split the attacks with AND-nodes and OR-nodes until you reach a simple enough level called elementary attacks.

Attack trees have been used to analyze real-life situations and many attack tree models have been proposed for that. However, only a few of the models allow us to do a quantified analysis of the security and compute the attacker's utility and most of them do not provide consistent results.

This thesis presented two multi-parameter attack tree models, that allow us to compute the attacker's expected utility of attack from the parameters associated with attack tree leaves. The parameters used to describe those leaves are the success probability of the attack, expected cost of the attack and associated penalties and also the gains of the whole attack tree. The proposed models are consistent, which means that when given two equivalent attack trees that describe the same attack, the computed utility value of both trees is equal. The first model, the parallel attack tree model, considers all the chosen attacks happening in parallel. The second model, the serial attack tree model, allows the attacker to choose the temporal order of the attacks and then adaptively decide if it makes sense to try the next attack or not. The serial attack tree model follows the real life situation more closely and also provides the greatest expected utility value.

The outcome computation in both models works by considering all satisfying attack suites, which drives the computation complexity exponential and prohibits the application of those models in the real life analysis with attack trees having more than 20-30 leaves. To work around those limitations, the thesis experimented with genetic algorithms. The performance and accuracy tests reveal that genetic algorithms are very usable and gave the correct outcome value in about 90% of the cases. They can process attack trees of up to 100 leaves, within 6 minutes.

Resümee (Abstract in Estonian)

Infoturbe mõõtmine on keeruline. Keeruline seetõttu, et mitme-astmeliste rünnete õnnestumise tõenäosust on raske hinnata ning tüüpiliselt ei ole teada, kui hästi rakendatud turvameetmed tegelikult turvalisust tagavad. Enamasti ei jaga rünnete ohvrid juba toimunud infoturbeintsidentide kohta andmeid ning seetõttu ei saa kasutada ka ajaloolistele andmetele ning sündmuste sagedusele tuginevat riskianalüüsi, et leida näiteks aasta kohta ootekahju.

Insenerid, kes peavad lahendama sarnast probleemi, kasutavad süsteemide töökindluse hindamiseks tõrkepuude analüüsi ning selgub, et samu printsiipe saab rakendada ka infoturbes. Vastavat metoodikat nimetatakse ründepuude analüüsiks. Ründepuude koostamisel analüüsitakse ründaja eesmärki ning jagatakse see eesmärk järk-järgult lihtsamateks rünneteks AND- ja OR-tippudega. AND-tipu õnnestumiseks peavad õnnestuma kõik selle tipu alamtipud ning OR-tipu õnnestumiseks peab õnnestuma vähemalt üks alamtipp. Viimasel tasemel olevaid puu lehti, mis on edasiseks kvantitatiivseks analüüsiks juba piisavalt lihtsad, nimetatakse elementaarrünneteks ning sedaviisi koostatakse keerukama ründe hierarhiline kirjeldus.

Peale ründepuude analüüsi loomist 1991. aastal on seda kasutatud paljude reaalsete situatsioonide analüüsimiseks ning selleks on pakutud mitmeid täpsustatud mudeleid. Mõned nendest sisaldavad ka arvutusreegleid infoturbe kvantitatiivseks mõõtmiseks ning võimaldavad leida ründaja ootetulu, kuid enamik nendest mudelitest ei ole kooskõlalised ning nende tulemused ei ole usaldusväärsed.

Doktoritöös esitatakse kahte uut autori poolt arendatud ründepuu arvutusmudelit, mis võimaldavad arvutada ründaja ootetulu, arvestades ründepuu lehtedega seotud sisendparameetreid. Parameetriteks kasutatakse elementaarründe õnnestumise tõenäosust, elementaarründe läbiviimise ootekulu ning kogu ründepuu õnnestumisel saadavat ootetulu. Kirjeldatud mudelid on kooskõlalised, mis tähendab, et ekvivalentsete ründepuude puhul, mis kirjeldavad semantiliselt sama

rünnet, arvutatakse puudele võrdne ründaja ootetulu.

Esimene mudel arvestab, et ründaja fikseerib enne ründama asumist kasutatavate elementaarrünnete komplekti ning kõik ründed teostatakse paralleelselt. Teine mudel käsitleb lisaks ka rünnete ajalist järjestust ning lubab ründajal teha rünnaku käigus otsuseid, kas proovida järgmisena järjekorras olevat rünnet, või kui selle õnnestumine ei mõjuta tema eesmärgi saavutamist, siis selle vahele jätta. Jadamudel modelleerib reaalsete ründajate käitumist täpsemini kui paralleelmudel ning juhul, kui kõigi elementaarrünnete kulu on positiivne, siis on jadamudeli abil arvutatud ründaja ootetulu suurem kui paralleelmudelis.

Mõlema mudeli arvutusreeglid kasutavad suurima ootetulu leidmiseks globaalset otsingut üle kõigi ründaja eesmärki realiseerivate ründekomplektide, mistõttu nende arvutuskeerukus on eksponentsiaalne ning ootetulu arvutamine on väga aeganõudev. 30 lehega ründepuude töötlemine toimub paralleelmudelis kümnekonna minutiga, kuid näiteks jadamudeli järgi ootetulu arvutamine on praktiliselt võimalik vaid kuni 13 lehega puude korral.

Igapäevastes turvaanalüüsides koostatud ründepuude suurus ulatub sadade lehtedeni ning selliste puude töötlemine ei ole täppisalgoritmidega enam võimalik. Doktoritöös katsetati ligilähedaste vastuste saamiseks geneetilisi algoritme. Selgus, et ligikaudu 90% kuni 95% juhtudel leiti geneetilise algoritmiga sama ootetulu väärtus kui täppisalgoritmiga ning sedaviisi õnnestub ligikaudu 6 minutiga töödelda ka ründepuid, milles on kuni 100 lehte.

Doktoritöös toodud ründepuude mudeleid saab seega rakendada praktilises turvaanalüüsis. Juhul, kui leitud ründaja ootetulu on positiivne, siis saame näidata, et ründajal on kasulik infosüsteemi rünnata ning seega on infosüsteemi turvalisus ebapiisav. Kui ründaja ootetulu on negatiivne, siis on võimalik, et süsteem on turvaline majanduslikult kaalutlevate ründajate vastu.

Vajalike turvameetrikate väärtuste usaldusväärne hindamine on üks olulisi valdkondi, milles tuleb uurimistööd jätkata. Vastavate lahenduste väljatöötamisel tuleb kindlasti teha koostööd muude eluvaldkondadega, nagu näiteks majandus, psühholoogia ja kriminalistika. Samuti on vajalik luua ootetulu veaarvutuse mudelid ning leida, millistes piirides võib parameetrite hindamisel eksida. Viimaks, ründepuude katsetamine reaalses elus ning korduvate eksperimentide tulemuste ennustamine võimaldaks kinnitada mudelite paikapidamist.

Appendix A

Curriculum Vitæ

1. Personal data

   Name: Aivo Jürgenson
   Date of Birth: 15.07.1978
   Place of Birth: Viljandi, Estonia
   Citizenship: Estonia

2. Contact information

   Postal address: P. Kerese 26-2, Tallinn, Estonia
   Phone: +372 50 77547
   E-mail: `aivo.jurgenson@eesti.ee`

3. Education

Educational institution	Graduation year	Education
Tallinn University of Technology	2004	BSc of informatics
Tallinn University of Technology	2006	MSc of informatics

4. Language competence

   Estonian language: high level (mother tongue)
   English language: high level

5. Special courses

Year	Course
2000	Middle-level management course
2001, 2003	NATO INFOSEC officer courses
2006	ITIL base course
2007	Microsoft, Defense in Depth: Securing Windows 2003 Server
2009	TOGAF 8 Enterprise Architect certificate

6. Professional employment

Period	Institution	Position
Jan 1996 – July 1996	C. R. Jakobson's Gymnasium	computer class administrator
Nov 1996 – Feb 2000	AS Matti	programmer
Sep 1997 – July 2000	Ministry of Foreign Affairs of Estonia	IT Department, communication technician
July 2000 – July 2003	Ministry of Foreign Affairs of Estonia	IT Department, data communication and information security bureau manager
July 2003 – Oct 2004	Ministry of Foreign Affairs of Estonia	IT Department, advisor
Oct 2004 – Nov 2005	Ministry of Foreign Affairs of Estonia	Diplomatic Security Department, information security manager
Nov 2005 – Jan 2010	Elion Enterprises Ltd	information security manager
Jan 2010 – . . .	Cybernetica AS	security engineer

7. Scientific work

Journal articles:

1. A. Andrusenko, A. Jürgenson, and J. Willemson. Serial model for attack tree computations. *KSII Transactions on Internet and Information Systems*, 2010. (*to appear*).

Conference proceedings:

1. A. Jürgenson and J. Willemson. Processing multi-parameter attack-trees with estimated parameter values. In A. Miyaji, H. Kikuchi, and K. Rannenberg, editors, *Advances in Information and Computer Security: Second International Workshop on Security (IWSEC 2007)*, volume 4752 of LNCS, pages 308–319. Springer-Verlag, 2007.
2. A. Buldas and A. Jürgenson. Does secure time-stamping imply collision-free hash functions? In W. Susilo, J. K. Liu, and Y. Mu, editors, *Proceedings of International Conference on Provable Security (ProvSec 2007)*, volume 4784 of LNCS, pages 138–150. Springer-Verlag, 2007.
3. A. Jürgenson and J. Willemson. Computing exact outcomes of multi-parameter attack trees. In R. Meersman and Z. Tari, editors, *On the Move to Meaningful Internet Systems (OTM 2008)*, volume 5332 of LNCS, pages 1036–1051. Springer-Verlag, 2008.
4. A. Buldas, A. Jürgenson, and M. Niitsoo. Efficiency bounds for adversary constructions in black-box reductions. In *ACISP '09: Proceedings of the 14th Australasian Conference on Information Security and Privacy*, pages 264–275. Springer-Verlag, 2009.
5. A. Jürgenson and J. Willemson. Serial model for attack tree computations. In D. Lee and S. Hong, editors, *Revised Papers from 12th International Conference on Information Security and Cryptology (ICISC 2009)*, volume 5984 of LNCS, pages 118–128. Springer-Verlag, 2010.
6. A. Jürgenson and J. Willemson. On fast and approximate attack tree computations. In J. Kwak, R. H. Deng, Y. Won, and G. Wang, editors, *Information Security Practice and Experience, 6th International Conference (ISPEC 2010)*, volume 6047 of LNCS, pages 56–66. Springer-Verlag, 2010.

Technical reports:

1. A. Jürgenson and J. Willemson. Ründepuud: pooladaptiivne mudel ja ligikaudsed arvutused (in Estonian). Technical Report T-4-4, Cybernetica, Institute of Information Security, 2009.

8. Defended theses

   1. BSc thesis (2004): "Security policy of an organization: Case study of the MFA of Estonia"
   2. MSc thesis (2006): "Risk analysis method based on attack trees with imprecise inputs"

9. Current research topics

Economic risk assessment models, attack trees, and information security in general.

Appendix B

Elulookirjeldus (CV in Estonian)

1. Isikuandmed

   Nimi: Aivo Jürgenson
   Sünniaeg: 15.07.1978
   Sünnikoht: Viljandi, Eesti
   Kodakondsus: Eesti

2. Sideandmed

   Postiaadress: P. Kerese 26-2, Tallinn, Eesti
   Telefon: +372 50 77547
   E-post: `aivo.jurgenson@eesti.ee`

3. Hariduskäik

Õppeasutus	Lõpetamise aeg	Haridus
Tallinna Tehnikaülikool	2004	informaatika, BSc
Tallinna Tehnikaülikool	2006	informaatika, MSc

4. Keelteoskus

   Eesti keel: kõrgtase (emakeel)
   Inglise keel: kõrgtase

5. Täiendusõpe

Aasta	Kursus
2000	Keskastme juhi kursus
2001, 2003	NATO INFOSEC ametniku kursused
2006	ITIL baaskursus
2007	Microsoft, Defense in Depth: Securing Windows 2003 Server
2009	TOGAF 8 Enterprise Architect sertifikaat

6. Teenistuskäik

Periood	Asutus	Ametikoht
jaan. 1996 – juuli 1996	C. R. Jakobsoni gümnaasium	arvutiklassi haldur
nov. 1996 – veeb. 2000	AS Matti	programmeerija
sept. 1997 – juuli 2000	Välisministeerium	IT osakond, sidetehnik
juuli 2000 – juuli 2003	Välisministeerium	IT osakond, andmeside- ja infoturbebüroo juhataja
juuli 2003 – okt. 2004	Välisministeerium	IT osakond, nõunik
okt. 2004 – nov. 2005	Välisministeerium	Diplomaatilise julgeoleku osakond, infoturbejuht
nov. 2005 – jaan. 2010	Elion Ettevõtted AS	infoturbejuht
jaan. 2010 – ...	Cybernetica AS	infoturbeinsener

7. Teadustegevus

Ajakirjaartiklid:

1. A. Andrusenko, A. Jürgenson, and J. Willemson. Serial model for attack tree computations. *KSII Transactions on Internet and Information Systems*, 2010. (*to appear*).

Konverentsid:

1. A. Jürgenson and J. Willemson. Processing multi-parameter attack-trees with estimated parameter values. In A. Miyaji, H. Kikuchi, and K. Rannenberg, editors, *Advances in Information and Computer Security: Second International Workshop on Security (IWSEC 2007)*, volume 4752 of LNCS, pages 308–319. Springer-Verlag, 2007.
2. A. Buldas and A. Jürgenson. Does secure time-stamping imply collision-free hash functions? In W. Susilo, J. K. Liu, and Y. Mu, editors, *Proceedings of International Conference on Provable Security (ProvSec 2007)*, volume 4784 of LNCS, pages 138–150. Springer-Verlag, 2007.
3. A. Jürgenson and J. Willemson. Computing exact outcomes of multi-parameter attack trees. In R. Meersman and Z. Tari, editors, *On the Move to Meaningful Internet Systems (OTM 2008)*, volume 5332 of LNCS, pages 1036–1051. Springer-Verlag, 2008.
4. A. Buldas, A. Jürgenson, and M. Niitsoo. Efficiency bounds for adversary constructions in black-box reductions. In *ACISP '09: Proceedings of the 14th Australasian Conference on Information Security and Privacy*, pages 264–275. Springer-Verlag, 2009.
5. A. Jürgenson and J. Willemson. Serial model for attack tree computations. In D. Lee and S. Hong, editors, *Revised Papers from 12th International Conference on Information Security and Cryptology (ICISC 2009)*, volume 5984 of LNCS, pages 118–128. Springer-Verlag, 2010.
6. A. Jürgenson and J. Willemson. On fast and approximate attack tree computations. In J. Kwak, R. H. Deng, Y. Won, and G. Wang, editors, *Information Security Practice and Experience, 6th International Conference (ISPEC 2010)*, volume 6047 of LNCS, pages 56–66. Springer-Verlag, 2010.

Tehnilised aruanded:

1. A. Jürgenson and J. Willemson. Ründepuud: pooladaptiivne mudel ja ligikaudsed arvutused (in Estonian). Technical Report T-4-4, Cybernetica, Institute of Information Security, 2009.

8. Kaitstud lõputööd

   1. Bakalaureusetöö (2004): "Asutuse turvapoliitika koostamine Välisministeeriumi näitel"
   2. Magistritöö (2006): "Ründepuudel põhinev riskianalüüsi meetod ebatäpsete sisendandmetega"

9. Teadustöö põhisuunad

Majandusliku riskianalüüsi mudelid, ründepuud ning üldine infoturve.

Appendix C

List of Publications

Three author's publications, which this thesis is based on, are included in the following pages:

Publication	Pages
A. Jürgenson and J. Willemson. Computing exact outcomes of multi-parameter attack trees. In R. Meersman and Z. Tari, editors, *On the Move to Meaningful Internet Systems (OTM 2008)*, volume 5332 of LNCS, pages 1036–1051. Springer-Verlag, 2008	111 – 129
A. Jürgenson and J. Willemson. Serial model for attack tree computations. In D. Lee and S. Hong, editors, *Revised Papers from 12th International Conference on Information Security and Cryptology (ICISC 2009)*, volume 5984 of LNCS, pages 118–128. Springer-Verlag, 2010	129 – 142
A. Jürgenson and J. Willemson. On fast and approximate attack tree computations. In J. Kwak, R. H. Deng, Y. Won, and G. Wang, editors, *Information Security Practice and Experience, 6th International Conference (ISPEC 2010)*, volume 6047 of LNCS, pages 56–66. Springer-Verlag, 2010	143 – 155

Computing Exact Outcomes of Multi-Parameter Attack Trees

Aivo Jürgenson[1,2] and Jan Willemson[3]

[1] Tallinn University of Technology, Raja 15, 12618 Tallinn, Estonia. aivo.jurgenson@eesti.ee

[2] Elion Enterprises Ltd, Endla 16, 15033 Tallinn, Estonia.

[3] Cybernetica, Aleksandri 8a, Tartu, Estonia. jan.willemson@gmail.com

Abstract In this paper we introduce a set of computation rules to determine the attacker's exact expected outcome based on a multi-parameter attack tree. We compare these rules to a previously proposed computational semantics by Buldas *et al.* and prove that our new semantics always provides at least the same outcome. A serious drawback of our proposed computations is the exponential complexity. Hence, implementation becomes an important issue. We propose several possible optimisations and evaluate the result experimentally. Finally, we also prove the consistency of our computations in the framework of Mauw and Oostdijk and discuss the need to extend the framework.

1 Introduction

Attack tree (also called threat tree) approach to security evaluation is several decades old. It has been used for tasks like fault assessment of critical systems [1] or software vulnerability analysis [2,3]. The approach was first applied in the context of information systems (so-called *threat logic trees*) by Weiss [4] and later more widely adapted to information security by Bruce Schneier [5]. We refer to [6,7] for good overviews on the development and applications of the methodology.

Even though already Weiss [4] realised that nodes of attack trees have many parameters in practise, several subsequent works in this field considered attack trees using only one estimated parameter like the cost or feasibility of the attack, skill level required, etc. [3,5,8]. Opel [9] considered also multi-parameter attack trees, but the actual tree computations in his model still used only one parameter at a time. Even though single-parameter attack trees can capture some aspects of threats reasonably well, they still lack the ability to describe the full complexity of the attacker's decision-making process.

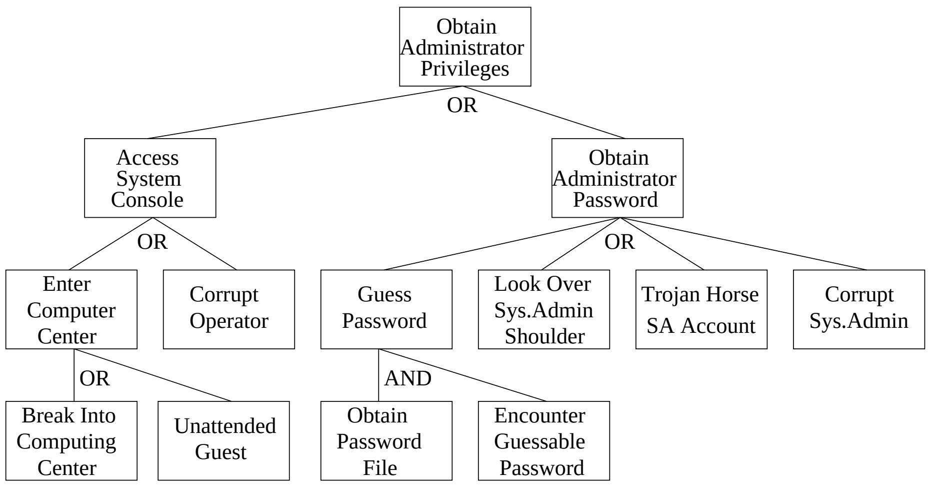
A substantial step towards better understanding the motivation of the attacker was made in 2006 by Buldas *et al.* [10]. Besides considering just the cost of the attack, they also used success probability together with probabilities and amount of penalties in the case of success or failure of the attack in their analysis. As a result, a more accurate model of the attack game was obtained and it was later used to analyse the security of several e-voting schemes by Buldas and Mägi [11]. The model was developed further by Jürgenson and Willemson [12] extending the parameter domain from point values to interval estimations.

However, it is known that the computational semantics given in [10] is both imprecise and inconsistent with the general framework introduced by Mauw and Oostdijk [8] (see Section 2). The motivation of the current paper is to develop a better semantics in terms of precision and consistency. For that we will first review the tree computations of [10] in Section 2 and then propose an improved semantics in Section 3. However, it turns out that the corresponding computational routines are inherently exponential, so optimisation issues of the implementation become important; these are discussed in Section 4. In Section 5 we prove that the new semantics always provides at least the same expected outcome for an attacker as the tree computations of [10]. We also argue that the new semantics is consistent with the framework of Mauw and Oostdijk. Finally, in Section 6 we draw some conclusions and set directions for further work.

2 Background

In order to better assess the security level of a complex and heterogeneous system, a gradual refinement method called *threat tree* or *attack tree method* can be used. The basic idea of the approach is simple — the analysis begins by identifying one or more *primary threats* and continues by splitting the threat into subattacks, either all or some of them being necessary to materialise the primary threat. The subattacks can be divided further etc., until we reach the state where it does not make sense to split the resulting attacks any more; these kinds of non-splittable attacks are called *elementary* or *atomic attacks* and the security analyst will have to evaluate them somehow. During the splitting process, a tree is formed having the primary threat in its root and elementary attacks in its leaves. Using the structure of the tree and the estimations of the leaves, it is then (hopefully) possible to give some estimations of the root node as well. In practise, it mostly turns out to be sufficient to consider only two kinds of splits in the internal nodes of the tree, giving rise to AND- and OR-nodes. As a result, an AND-OR-tree is obtained, forming the basis of the subsequent analysis. An example attack tree originally given by Weiss [4] and adopted from [6] is presented in Figure 1.

Figure 1. Example of an attack tree



We will use the basic multi-parameter attack tree model introduced in [10]. Let us have the AND-OR-tree describing the attacks and assume all the elementary attacks being pairwise independent. Let each leaf X_i have the following parameters:

- Cost_i – the cost of the elementary attack
- p_i – success probability of the attack
- π_i^- – the expected penalty in case the attack was unsuccessful
- π_i^+ – the expected penalty in case the attack was successful

Besides these parameters, the tree has a global parameter Gains showing the benefit of the attacker in the case he is able to mount the root attack. For practical examples on how to evaluate those parameters for real-life attacks, please refer to [11] and [13].

The paper [10] gives a simple computational semantics to the attack trees, which has further been extended to interval estimates in [12]. After the above-mentioned parameters have been estimated for the leaf nodes, a step-by-step propagation algorithm begins computing the same parameters for all the internal nodes as well, until the root node has been reached. The computational routines defined in [10] are the following:

- For an OR-node with child nodes with parameters $(\mathsf{Cost}_i, p_i, \pi_i^+, \pi_i^-)$ $(i = 1, 2)$ the parameters $(\mathsf{Cost}, p, \pi^+, \pi^-)$ are computed as:

$$(\mathsf{Cost}, p, \pi^+, \pi^-) = \begin{cases} (\mathsf{Cost}_1, p_1, \pi_1^+, \pi_1^-), \text{ if } \mathsf{Outcome}_1 > \mathsf{Outcome}_2 \\ (\mathsf{Cost}_2, p_2, \pi_2^+, \pi_2^-), \text{ if } \mathsf{Outcome}_1 \leq \mathsf{Outcome}_2 \end{cases},$$

$$\mathsf{Outcome}_i = p_i \cdot \mathsf{Gains} - \mathsf{Cost}_i - p_i \cdot \pi_i^+ - (1 - p_i) \cdot \pi_i^- .$$

- For an AND-node with child nodes with parameters $(\mathsf{Cost}_i, p_i, \pi_i^+, \pi_i^-)$ $(i = 1, 2)$ the parameters $(\mathsf{Cost}, p, \pi^+, \pi^-)$ are computed as follows:

$$\mathsf{Costs} = \mathsf{Costs}_1 + \mathsf{Costs}_2, \quad p = p_1 \cdot p_2, \quad \pi^+ = \pi_1^+ + \pi_2^+,$$

$$\pi^- = \frac{p_1(1-p_2)(\pi_1^+ + \pi_2^-) + (1-p_1)p_2(\pi_1^- + \pi_2^+)}{1 - p_1 p_2} + \\ + \frac{(1-p_1)(1-p_2)(\pi_1^- + \pi_2^-)}{1 - p_1 p_2} .$$

  The formula for π^- represents the average penalty of an attacker, assuming that at least one of the two child-attacks was not successful. For later computations, it will be convenient to denote expected expenses associated with the node i as $\mathsf{Expenses}_i = \mathsf{Cost}_i + p_i \cdot \pi_i^+ + (1-p_i) \cdot \pi_i^-$. Then it is easy to see that in an AND-node the equality $\mathsf{Expenses} = \mathsf{Expenses}_1 + \mathsf{Expenses}_2$ holds. Note that the formulae above have obvious generalisations for non-binary trees.

At the root node, its $\mathsf{Outcome}$ is taken to be the final outcome of the attack and the whole tree is considered to be beneficial for a rational attacker if $\mathsf{Outcome} > 0$. Following the computation process it is possible to collect the corresponding set of leaves which, when carried out, allow the attacker to mount the root attack and get the predicted outcome. Such leaf sets will subsequently be called *attack suites*.[4]

However, while being very fast to compute, this semantics has several drawbacks:

1. In order to take a decision in an OR-node, the computational model of [10] needs to compare outcomes of the child nodes and for that some local estimate of the obtained benefit is required. Since it is very difficult to break the total root gain into smaller benefits, the model of [10] gives the total amount of Gains to the attacker for each subattack. This is clearly an overestimation of the attacker's outcome.

[4] Note that our terminology differs here from the one used by Mauw and Oostdijk [8]. Our attack suite would be just attack in their terms and their attack suite would be the set of all possible attack suites for us.

2. In an OR-node, the model of [10] assumes that the attacker picks exactly one descendant. However, it is clear that in practise, it may make sense for an attacker to actually carry out several alternatives if the associated risks and penalties are low and the success probability is high.
3. There is a general result by Mauw and Oostdijk [8] stating which attack tree computation semantics are inherently consistent. More precisely, they require that the semantics of the tree should remain unchanged when the underlying Boolean formula is transformed to an equivalent one (e.g. to a disjunctive normal form). Semantics given in the [10] are not consistent in this sense. For example, lets take two attack trees, $T_1 = A \vee (B\&C)$ and $T_2 = (A \vee B)\&(A \vee C)$, both having same parameters $\mathsf{Gains} = 10000$, $p_A = 0.1$, $p_B = 0.5$, $p_C = 0.4$, $\mathsf{Expenses}_A = 1000$, $\mathsf{Expenses}_B = 1500$, $\mathsf{Expenses}_C = 1000$. Following the computation rules of [10], we get $\mathsf{Outcome}_{\mathsf{T}_1} = 8000$ and $\mathsf{Outcome}_{\mathsf{T}_2} = 6100$, even though the underlying Boolean formulae are equivalent.

The aim of this paper is to present an exact and consistent semantics for attack trees. The improved semantics fixes all the three abovementioned shortcomings. However, a major drawback of the new approach is the increase of the computational complexity from linear to exponential (depending on the number of elementary attacks). Thus finding efficient and good approximations becomes a vital task. In this paper, we will evaluate suitability of the model of [10] as an approximation; the question of better efficient approximations remains an open problem for future research.

3 Exact Semantics for the Attack Trees

3.1 The model

In our model, the attacker behaves as follows.

– First, the attacker constructs an attack tree and evaluates the parameters of its leaves.
– Second, he considers all the potential attack suites, i.e. subsets $\sigma \subseteq \mathcal{X} = \{X_i : i = 1, \ldots, n\}$. Some of these materialise the root attack, some of them do not. For the suites that do materialise the root attack, the attacker evaluates their outcome for him.
– Last, the attacker decides to mount the attack suite with the highest outcome (or he may decide not to attack at all if all the outcomes are negative).

Note that in this model the attacker tries all the elementary attacks independently. In practise, this is not always true. For example, if the attacker has already failed some critical subset of the suite, it may make more sense for him not to try the rest of the suite. However, the current model is much more realistic compared to the one described in [10], since now we allow the attacker to plan its actions with redundancy, i.e. try alternative approaches to achieve some (sub)goal.

3.2 Formalisation

The attack tree can be viewed as a Boolean formula $\mathcal{F}$ composed of the set of variables $\mathcal{X} = \{X_i : i = 1, \dots, n\}$ (corresponding to the elementary attacks) and conjunctives $\vee$ and &. Satisfying assignments $\sigma \subseteq \mathcal{X}$ of this formula correspond to the attack suites sufficient for materialising the root attack.

The exact outcome of the attacker can be computed as

$$\mathsf{Outcome} = \max\{\mathsf{Outcome}_\sigma : \sigma \subseteq \mathcal{X},\, \mathcal{F}(\sigma := \mathsf{true}) = \mathsf{true}\}\,. \qquad (1)$$

Here $\mathsf{Outcome}_\sigma$ denotes the expected outcome of the attacker if he decides to try the attack suite σ and $\mathcal{F}(\sigma := \mathsf{true})$ denotes evaluation of the formula $\mathcal{F}$, when all of the variables of σ are assigned the value true and all others the value false. The expected outcome $\mathsf{Outcome}_\sigma$ of the suite σ is computed as follows:

$$\mathsf{Outcome}_\sigma = p_\sigma \cdot \mathsf{Gains} - \sum_{X_i \in \sigma} \mathsf{Expenses}_i\,, \qquad (2)$$

where p_σ is the success probability of the attack suite σ.

When computing the success probability p_σ of the attack suite σ we must take into account that the suite may contain redundancy and there may be (proper) subsets $\rho \subseteq \sigma$ sufficient for materialising the root attack. Because we are using the full suite of σ to mount an attack, those elementary attacks in the $\sigma \setminus \rho$ will contribute to the success probability of p_ρ with $(1 - p_j)$. Thus, the total success probability can be computed as

$$p_\sigma = \sum_{\substack{\rho \subseteq \sigma \\ \mathcal{F}(\rho := \mathsf{true}) = \mathsf{true}}} \prod_{X_i \in \rho} p_i \prod_{X_j \in \sigma \setminus \rho} (1 - p_j)\,. \qquad (3)$$

Note that the formulae (1), (2) and (3) do not really depend on the actual form of the underlying formula $\mathcal{F}$, but use it only as a Boolean function. As a consequence, our framework is not limited to just AND-OR trees, but can in principle accommodate other connectives as well. Independence of the concrete form will also be the key observation when proving the consistency of our computation routines in the framework of Mauw and Oostdijk (see Proposition 1 in Section 5).

3.3 Example

To explain the exact semantics model of the attack trees, we give the following simple example. Lets consider the attacktree with the Boolean formula $T = (A \vee B)\&C$ with all elementary attacks (A, B, C) having equal parameters $p = 0.8$, $\mathsf{Cost} = 100$, $\pi^+ = 1000$, $\pi^- = 1000$ and $\mathsf{Gain} = 10000$. That makes $\mathsf{Expenses} = 1100$ for all elementary attacks. When we follow the approximate computation rules in the [10], we get the $\mathsf{Outcome}_T = 4200$.

By following the computation rules in this article, we have the attack suites $\sigma_1 = \{A, C\}, \sigma_2 = \{B, C\}, \sigma_3 = \{A, B, C\}$, which satisfy the original attack tree T. The outcome computation for attack suites σ_1 and σ_2 is straightforward and $\mathsf{Outcome}_{\sigma_1} = \mathsf{Outcome}_{\sigma_2} = 4200$. The $\mathsf{Outcome}_{\sigma_3}$ is a bit more complicated as there are three subsets $\rho_1 = \{A, C\}$, $\rho_2 = \{B, C\}$, $\rho_3 = \{A, B, C\}$ for the suite σ_3, which also satisfy the attack tree T. Therefore we get the $p_{\sigma_3} = p_A p_B p_C + p_A p_C(1-p_B) + p_B p_C(1-p_A) = 0.768$ and $\mathsf{Outcome}_{\sigma_3} = 4380$. By taking the maximum of the three outcomes, we get $\mathsf{Outcome}_T = 4380$.

As the Cost parameters in this example for elementary attacks A and B were chosen quite low and the success probability p_A and p_B of these attacks were quite high, it made sense for an attacker to mount both of these subattacks and get bigger expected outcome, even though the attack tree would have been satisfied as well by only one of them.

4 Implementation

The most time-consuming computational routine among the computations given in Section 3.2 is the generation of all the satisfiable assignments of a Boolean formula $\mathcal{F}$ in order to find the maximal outcome by (1). Even though the computation routine (3) for finding p_σ formally also goes through (potentially all) subsets of σ, it can be evaluated in linear time in the number of variables n. To do so we can set $p_i = 0$ for all $X_i \notin \sigma$ and leave all the p_i for $X_i \in \sigma$ untouched. Then for each internal node of the tree with probabilities of the child nodes being $p_{i_1}, p_{i_2}, \ldots, p_{i_k}$

we can compute the probability of the parent node to be

$$\prod_{j=1}^{k} p_{i_j} \quad \text{or} \quad 1 - \prod_{j=1}^{k}(1 - p_{i_j})$$

depending on whether it is an AND or an OR node. Propagating throughout the tree, this computation gives exactly the success probability p_σ of the suite σ at the root node.

The routine (1) can be optimised as well by cutting off hopeless cases (see Theorem 1), but it still remains worst-case exponential-time. Thus for performance reasons it is crucial to have an efficient implementation of this routine. We are using a modified version of DPLL algorithm [14] to achieve this goal. The original form of the DPLL algorithm is only concerned about satisfiability, but it can easily be upgraded to produce all the satisfying assignments as well. Note that all the assignments are not needed at the same time to compute (1), but rather one at a time. Hence we can prevent the exponential memory consumption by building a serialised version, obtaining Algorithm 1.

Algorithm 1 works recursively and besides the current Boolean formula $\mathcal{F}$ it has two additional parameters. The set S contains the variables of which the satisfying assignments should be composed from. The set A on the other hand contains the variables already chosen to the assignments on previous rounds of recursion. As a technical detail note that the satisfying assignments are identified by the subset of variables they set to true.

The computation starts by calling process_satisfying_assignments($\mathcal{F}$, $\mathcal{X}$, $\emptyset$). Note that Algorithm 1 does not really produce any output, a processing subroutine is called on step 1 instead. This subroutine computes $\mathsf{Outcome}_\sigma$ for the given assignment σ and compares it with the previous maximal outcome.

4.1 Optimisations

Even with the help of a DPLL-based algorithm, the computations of (1) remain worst-case exponential time. In order to cut off hopeless branches, we can make some useful observations.

When we consider a potential attack suite σ and want to know, whether it is sufficient to materialise the root attack, we will set all the elements of σ to true, all the others to false and evaluate the formula $\mathcal{F}$ corresponding to the attack tree. In the process, all the internal nodes of the tree get evaluated as well (including the root node, showing whether the suite is

Algorithm 1 Processing all the satisfying assignments of a formula

```
Procedure process_satisfying_assignments($\mathcal{F}, S, A$)
Input: Boolean CNF-formula $\mathcal{F}$, a subsets $S$ of its variables and a subset $A \subseteq \mathcal{X} \setminus S$

 1. If $\mathcal{F}$ contains true in every clause then
      – Process the assignment $A \cup T$ for every $T \subseteq S$; return
 2. If $\mathcal{F}$ contains an empty clause or $S = \emptyset$ then return #no output in this branch
 3. If $\mathcal{F}$ contains a unit clause $\{X\}$, where $X \in S$ then
      – Let $\mathcal{F}'$ be the formula obtained by setting $X =$ true in $\mathcal{F}$
      – process_satisfying_assignments($\mathcal{F}', S \setminus \{X\}, A \cup \{X\}$)
      – Return
 4. Select a variable $X \in S$
 5. Let $\mathcal{F}'$ be the formula obtained by setting $X =$ true in $\mathcal{F}$
 6. process_satisfying_assignments($\mathcal{F}', S \setminus \{X\}, A \cup \{X\}$)
 7. Let $\mathcal{F}''$ be the formula obtained by deleting $X$ from $\mathcal{F}$
 8. process_satisfying_assignments($\mathcal{F}'', S \setminus \{X\}, A$)
 9. Return
```

sufficient). In Section 3, we allowed the suites σ to have more elements than absolutely necessary for materialising the root node, because in OR-nodes it often makes a lot of sense to try different alternatives. In AND nodes, at the same time, no choice is actually needed and achieving some children of an AND node without achieving some others is just a waste of resources.

Thus, intuitively we can say that it makes no sense to have AND-nodes with some children evaluating to true and some children to false. Formally, we can state and prove the following theorem.

Theorem 1. *Let $\mathcal{F}$ be a Boolean formula corresponding to the attack tree T (i.e. AND-OR-tree, where all variables occur only once) and let σ be its satisfying assignment (i.e. an attack suite). Set all the variables of σ to* true *and all others to* false *and evaluate all the internal nodes of T. If some AND-node has children evaluating to* true *as well as children evaluating to* false*, then there exists a satisfying assignment $\sigma' \subset \sigma$ ($\sigma' \neq \sigma$) such that $\mathsf{Outcome}_{\sigma'} \geq \mathsf{Outcome}_{\sigma}$.*

Proof. Consider an AND-node Y having some children evaluating to true and some evaluating to false. Then the node Y itself also evaluates to false, but the set of variables of the subformula corresponding to Y has a non-empty intersection with σ; let this intersection be τ. We claim that we can take $\sigma' = \sigma \setminus \tau$. First it is clear that $\sigma' \subset \sigma$ and $\sigma' \neq \sigma$. Note also that σ' is a satisfying assignment and hence $\sigma' \neq \emptyset$. Now consider the corresponding outcomes:

$$\begin{aligned}\mathsf{Outcome}_\sigma &= p_\sigma \cdot \mathsf{Gains} - \sum_{X_i \in \sigma} \mathsf{Expenses}_i\,,\\ \mathsf{Outcome}_{\sigma'} &= p_{\sigma'} \cdot \mathsf{Gains} - \sum_{X_i \in \sigma'} \mathsf{Expenses}_i\,.\end{aligned}$$

Since $\sigma' \subset \sigma$, we have

$$\sum_{X_i \in \sigma} \mathsf{Expenses}_i \geq \sum_{X_i \in \sigma'} \mathsf{Expenses}_i\,,$$

as all the added terms are non-negative.

Now we claim that the equality $p_\sigma = p_{\sigma'}$ holds, which implies the claim of the theorem. Let

$$R_\sigma = \{\rho \subseteq \sigma \,:\, \mathcal{F}(\rho := \mathsf{true}) = \mathsf{true}\}$$

and define $R_{\sigma'}$ in a similar way. Then by (3) we have

$$\begin{aligned}p_\sigma &= \sum_{\rho \in R_\sigma} \prod_{X_i \in \rho} p_i \prod_{X_j \in \sigma \setminus \rho} (1 - p_j)\,,\\ p_{\sigma'} &= \sum_{\rho' \in R_{\sigma'}} \prod_{X_i \in \rho'} p_i \prod_{X_j \in \sigma' \setminus \rho'} (1 - p_j)\,.\end{aligned}$$

We claim that $R_\sigma = \{\rho' \cup \tau' \,:\, \rho' \in R_{\sigma'}, \tau' \subseteq \tau\}$, i.e. that all the satisfying subassignments of σ can be found by adding all the subsets of τ to all the satisfying subassignments of σ'. Indeed, the node Y evaluates to false even if all the variables of τ are true, hence the same holds for every subset of τ due to monotonicity of AND and OR. Thus, if a subassignment of σ satisfies the formula $\mathcal{F}$, the variables of τ are of no help and can have arbitrary values. The evaluation true for the root node can only come from the variables of σ', proving the claim.

Now we can compute:

$$\begin{aligned}p_\sigma = \sum_{\rho \in R_\sigma} \prod_{X_i \in \rho} p_i \prod_{X_j \in \sigma \setminus \rho} (1 - p_j) = &\sum_{\substack{\rho = \rho' \cup \tau' \\ \rho' \in R_{\sigma'}, \tau' \subseteq \tau}} \prod_{X_i \in \rho} p_i \prod_{X_j \in \sigma \setminus \rho} (1 - p_j) =\\ = \sum_{\rho' \in R_{\sigma'}} \sum_{\tau' \subseteq \tau} \prod_{X_i \in \rho' \cup \tau'} p_i \prod_{X_j \in \sigma \setminus (\rho' \cup \tau')} (1 - p_j) =\end{aligned}$$

$$
\begin{aligned}
&= \sum_{\rho'\in R_{\sigma'}} \sum_{\tau'\subseteq\tau} \prod_{X_i\in\rho'} p_i \prod_{X_i\in\tau'} p_i \prod_{X_j\in\sigma'\setminus\rho'} (1-p_j) \prod_{X_j\in\tau\setminus\tau'} (1-p_j) = \\
&= \sum_{\rho'\in R_{\sigma'}} \prod_{X_i\in\rho'} p_i \prod_{X_j\in\sigma'\setminus\rho'} (1-p_j) \sum_{\tau'\subseteq\tau} \prod_{X_i\in\tau'} p_i \prod_{X_j\in\tau\setminus\tau'} (1-p_j) = \\
&= \sum_{\rho'\in R_{\sigma'}} \prod_{X_i\in\rho'} p_i \prod_{X_j\in\sigma'\setminus\rho'} (1-p_j) \prod_{X_i\in\tau} [p_i + (1-p_i)] = \\
&= \sum_{\rho'\in R_{\sigma'}} \prod_{X_i\in\rho'} p_i \prod_{X_j\in\sigma'\setminus\rho'} (1-p_j) = p_{\sigma'} \,,
\end{aligned}
$$

since $\sigma\setminus(\rho'\cup\tau') = (\sigma'\setminus\rho')\dot{\cup}(\tau\setminus\tau')$. The claim of the theorem now follows easily. □

Note that Theorem 1 really depends on the assumption that $\mathcal{F}$ is an AND-OR-tree and that all variables occur only once. Formulae (1) and (3) together with Algorithm 1 can still be applied if the structure of the formula $\mathcal{F}$ is more complicated (say, a general DAG with other connectives in internal nodes), but the optimisation of Theorem 1 does not necessarily work.

This theorem allows us to leave many potential attack suites out of consideration by simply verifying if they evaluate children of some AND-node in a different way.

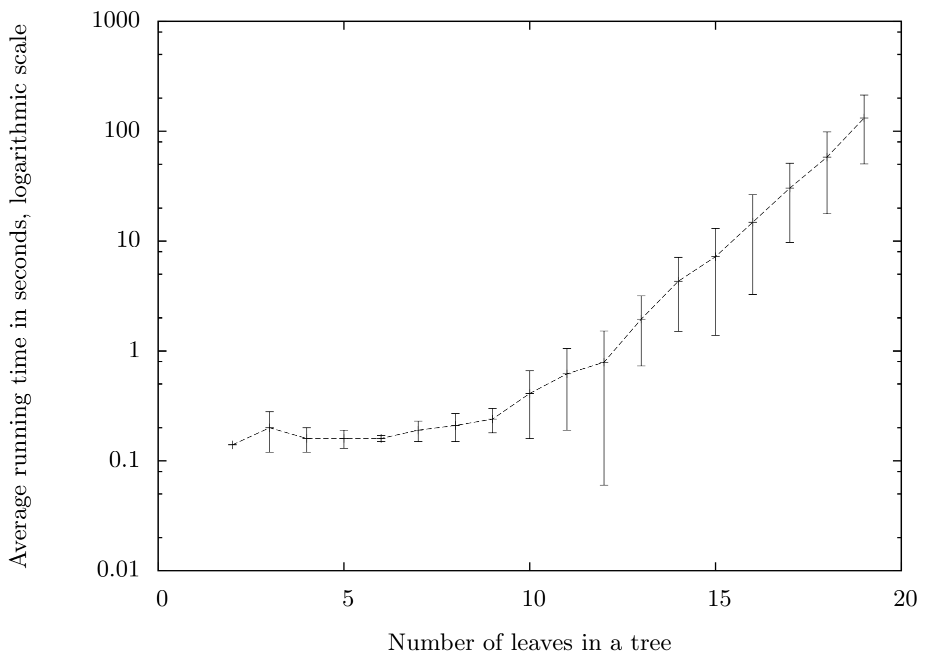
4.2 Performance

We implemented Algorithm 1 in Perl programming language and ran it on 500 randomly generated trees. The tests were ran on a computer having 3GHz dual-core Intel processor, 1GB of RAM and Arch Linux operating system.

The tree generation procedure was the following:

1. Generate the root node.
2. With probability 50% let this node have 2 children and with probability 50% let it have 3 children.
3. For every child, let it be an AND-node, an OR-node or a leaf with probability 40%, 40% and 20%, respectively.
4. Repeat the steps number 2 and 3 for every non-leaf node until the tree of depth up to 3 has been generated and let all the nodes on the third level be leaves.
5. To all the leaf nodes, generate the values of Cost, π^+ and π^- as integers chosen uniformly from from the interval $[0, 1000)$, and the value of p chosen uniformly from the interval $[0, 1)$.

Figure 2. Performance test results



6. Generate the value of Gains as an integer chosen uniformly from the interval $[0, 1000000)$.

Thus, the generated trees may in theory have up to 27 leaves. That particular size limit for the trees was chosen because the running time for larger trees was already too long for significant amount of tests.

Performance test results showing the average running times and the standard deviation of the running times of the algorithm depending on the number of leaves are displayed in Figure 2. Note that the time scale is logarithmic. The times are measured together with the conversion of the attack tree formula to the conjunctive normal form. In Figure 2 we have included the trees with only up to 19 leaves, since the number of larger trees generated was not sufficient to produce statistically meaningful results. The number of the generated trees by the number of leaves is given later in Figure 3.

5 Analysis

In this Section we provide some evaluation of our tree computations compared to the ones given by Buldas *et al.* [10] and within the framework of Mauw and Oostdijk [8].

5.1 Comparison with the semantics of Buldas *et al.*

Our main result can be shortly formulated as the following theorem.

Theorem 2. *Let us have an attack tree T. Let the best attack suites found by the routines of the current paper and the paper [10] be σ and σ' respectively. Let the corresponding outcomes (computed using the respective routines) be $\mathsf{Outcome}_\sigma$ and $\mathsf{Outcome}_{\sigma'}$. The following claims hold:*

1. If $\sigma = \sigma'$ then $\mathsf{Outcome}_\sigma = \mathsf{Outcome}_{\sigma'}$.
2. $\mathsf{Outcome}_\sigma \geq \mathsf{Outcome}_{\sigma'}$.

Proof.

1. We need to prove that if $\sigma = \sigma'$ then

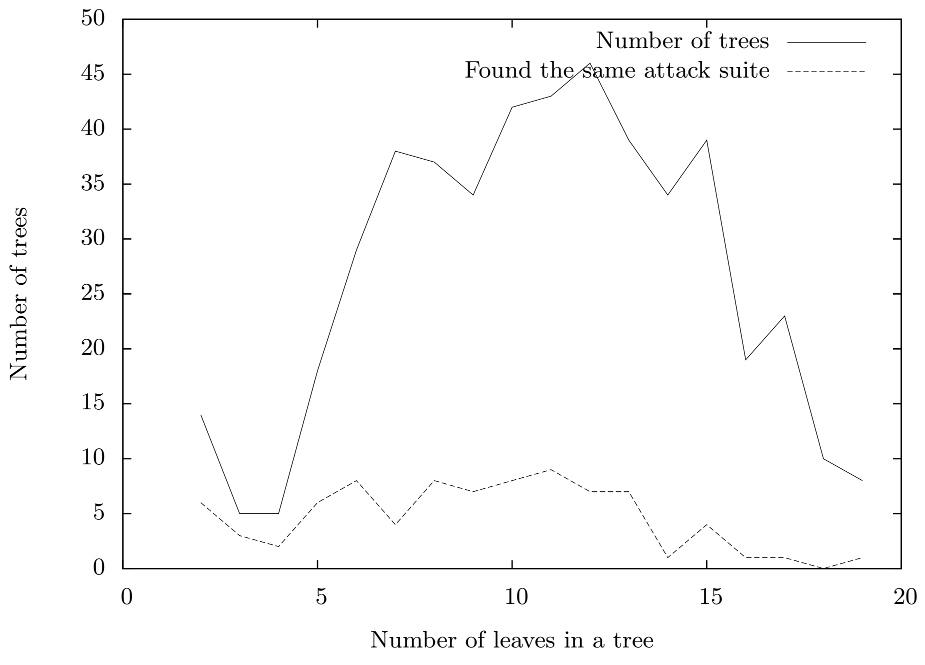
$$\mathsf{Outcome}_{\sigma'} = p_\sigma \cdot \mathsf{Gains} - \sum_{X_i \in \sigma} \mathsf{Expenses}_i \,.$$

   First note that the attack suite output by the routine of [10] is minimal in the sense that none of its proper subsets materialises the root node, because only one child is chosen in every OR-node. Hence, $p_\sigma = \prod_{X_i \in \sigma} p_i$. Now consider how $\mathsf{Outcome}_{\sigma'}$ of the root node is computed in [10]. Let the required parameters of the root node be p', Gains' and $\mathsf{Expenses}'$. Obviously, $\mathsf{Gains}' = \mathsf{Gains}$. By looking at how the values of the attack success probability and the expected expenses are propagated throughout the tree, we can also conclude that

$$p' = \prod_{X_i \in \sigma} p_i = p_\sigma \quad \text{and} \quad \mathsf{Expenses}' = \sum_{X_i \in \sigma} \mathsf{Expenses}_i \,,$$

   finishing the first part of the proof.
2. Since σ' is a satisfying assignment of the Boolean formula underlying the tree T, we can conclude that σ' is considered as one of the attack suite candidates in (1). The conclusion now follows directly from the first part of the proof. □

Figure 3. Precision of the computational routine of Buldas *et al.* [10]



Theorem 2 implies that the exact attack tree computations introduced in the current paper always yield at least the same outcome compared to [10]. Thus, the potential use of the routine of [10] is rather limited, because it only allows us to get a lower estimate of the attacker's expected outcome, whereas the upper limit would be of much higher interest. We can still say that if the tree computations of [10] show that the system is insufficiently protected (i.e. $\mathsf{Outcome}_{\sigma'} > 0$) then the exact computations would yield a similar result ($\mathsf{Outcome}_{\sigma} > 0$).

Following the proof of Theorem 2, we can also see that the semantics of [10] is actually not too special. Any routine that selects just one child of every OR-node when analysing the tree would essentially give a similar under-estimation of the attacker's expected outcome.

Together with the performance experiments described in Section 4.2 we also compared the outcome attack suites produced by the routines of the current paper and [10] (the implementation of the computations of [10] was kindly provided by Alexander Andrusenko [15]). The results are depicted in Figure 3.

The graphs in Figure 3 show the number of the generated trees by the number of leaves and the number of such trees among them, for which the routine of [10] was able to find the same attack suite that the exact computations introduced in the current paper. Over all the tests we can say that this was the case with 17.4% of the trees.

5.2 Consistency with the framework of Mauw and Oostdijk

Working in a single parameter model, Mauw and Oostdijk [8] first define a set V of attribute values and then consider an attribute function $\alpha : \mathbb{C} \to V$, where $\mathbb{C}$ is the set of elementary attacks (called *attack components* in [8]). In order to extend this attribution to the whole tree, they essentially consider the tree corresponding to the disjunctive normal form of the underlying Boolean formula. To obtain the attribute values of the conjunctive clauses (corresponding to our attack suites), they require a conjunctive combinator $\triangle : V \times V \to V$, and in order to get the value for the whole DNF-tree based on clause values they require a disjunctive combinator $\bigtriangledown : V \times V \to V$. Mauw and Oostdijk prove that if these combinators are commutative, associative and distributive, all the nodes of the tree in the original form can also be given attribute values and that the value of the (root node of the) tree does not change if the tree is transformed into an equivalent form. This equivalence is denoted as $\equiv$ and it is defined by the set of legal transformations retaining logical equivalence of the underlying Boolean formulae (see [8]). The structure $(\alpha, \bigtriangledown, \triangle)$ satisfying all the given conditions is called *distributive attribute domain.*

Even though the semantics used in [10,12] formally require four different parameters, they still fit into a single parameter ideology, since based on the quadruples of the child nodes, similar quadruples are computed for parents when processing the trees. However, it is easy to construct simple counterexamples showing that the computation rules of [10,12] are not distributive, one is given in the Section 2.

The computation rules presented in the current paper follow the framework of Mauw and Oostdijk quite well at the first sight. Formula (1) essentially goes through all the clauses in the complete disjunctive normal form of the underlying formula $\mathcal{F}$ and finds the one with the maximal outcome. So we can take $V = \mathbb{R}$ and $\bigtriangledown = \max$ in the Mauw and Oostdijk framework. However, there is no reasonable way to define a conjunctive combinator $\triangle : V \times V \to V$, since the outcome of an attack suite can not be computed from the outcomes of the elementary attacks; the phrase "outcome of an elementary attack" does not even have a meaning.

Another possible approach is to take $V = [0,1] \times \mathbb{R}^+$ and to interpret the first element of $\alpha(X)$ as the success probability p and the second element as $\mathsf{Expenses}$ for an attack X. Then the disjunctive combinator can be defined as outputting the pair which maximises the expression $p \cdot \mathsf{Gains} - \mathsf{Expenses}$. This combinator has a meaning in the binary case and as such, it is both associative and commutative, giving rise to an obvious n-ary generalisation. For the conjunctive combinator to work as expected in the n-ary case, we would need to achieve

$$\triangle_{X_i \in \sigma} \alpha(X_i) = (p_\sigma, \Sigma_{X_i \in \sigma} \mathsf{Expenses}_i)\,.$$

However, it is easy to construct a formula and a satisfying assignment σ such that constructing p_σ from success probabilities of the descendant instances using a conjunctive combinator is not possible. For example, we can take the formula $\mathcal{F} = X_1 \vee X_2 \& X_3$, where X_1, X_2, X_3 are elementary attacks with success probabilities p_1, p_2, p_3, respectively. Let α_1 denote the first element of the output of α and let $\triangle_1$ denote the combinator $\triangle$ restricted to the first element of the pair (so $\triangle_1(X_i) = p_i$, $i = 1, 2, 3$). Then for $\sigma = \{X_1, X_2, X_3\}$ we would need to obtain

$$(p_1 \triangle_1 p_2) \triangle_1 p_3 = (\alpha_1(X_1) \triangle_1 \alpha_1(X_2)) \triangle_1 \alpha_1(X_3) = p_\sigma = p_1 + p_2 p_3 - p_1 p_2 p_3$$

for any $p_1, p_2, p_3 \in [0,1]$, which is not possible. Indeed, taking $p_3 = 0$ we have $(p_1 \triangle_1 p_2) \triangle_1 0 = p_1$. In the same way we can show that $(p_2 \triangle_1 p_1) \triangle_1 0 = p_2$, which is impossible due to commutativity of $\triangle_1$ when $p_1 \neq p_2$.

All of the above is not a formal proof that our computations do not form a distributive attribute domain, but we can argue that there is no obvious way to interpret them as such. Additionally, if we had a distributive attribute domain then Theorem 3 with Corollary 2 of [8] would allow us to build a linear-time value-propagating tree computation algorithm, but this is rather unlikely.

However, we can still state and prove the following proposition.

Proposition 1. *Let T_1 and T_2 be two attack trees. If $T_1 \equiv T_2$, we have $\mathsf{Outcome}(T_1) = \mathsf{Outcome}(T_2)$.*

Proof. It is easy to see that the formulae (1), (2) and (3) do not depend on the particular form of the formula, but use it only as a Boolean function. Since the tree transformations defined in [8] keep the underlying Boolean formula logically equivalent, the result follows directly. □

In the context of [8], this is a somewhat surprising result. Even though the attribute domain defined in the current paper is not distributive (and it can not be easily turned into such), the main goal of Mauw and Oostdijk is still achieved. This means that the requirement for the attribute domain to be distributive in the sense of Mauw and Oostdijk is sufficient to have semantically consistent tree computations, but it is not really necessary. It would be interesting to study, whether the framework of Mauw and Oostdijk can be generalised to cover non-propagating tree computations (like the one presented in the current paper) as well.

6 Conclusions and Further Work

In this paper we introduced a computational routine capable of finding the maximal possible expected outcome of an attacker based on a given attack tree. We showed that when compared to rough computations given in [10], the new routine always gives at least the same outcome and mostly it is also strictly larger. This means that the tree computations of [10] are not very useful in practise, since they strongly tend to under-estimate attacker's capabilities. We also proved that unlike [10], our new semantics of the attack tree is consistent with the general ideology of the framework of Mauw and Oostdijk, even though our attribute domain is not distributive. This is a good motivation to start looking for further generalisations of the framework.

On the other hand, the routines of the current paper are computationally very expensive and do not allow practical analysis of trees with the number of leaves substantially larger than 20. Thus, future research needs to address at least two issues. First, there are some optimisations possible in the implementation (e.g. precomputation of frequently needed values), they need to be programmed and compared to the existing implementations. Still, any optimisation will very probably not decrease the time complexity of the algorithm to a subexponential class. Thus the second direction of further research is finding computationally cheap approximations, which would over-estimate the attacker's exact outcome.

As a further development of the attack tree approach, more general and realistic models can be introduced. For example, the model presented in the current paper does not take into account the possibility that the attacker may drop attempting an attack suite after a critical subset of it has already failed. Studying such models will remain the subject for future research as well.

Acknowledgments

This research has been supported by the Estonian Science Foundation grant no. 7081. Also, we would like to thank Ahto Buldas, Peeter Laud and Sven Laur for helpful discussions.

References

1. W.E. Vesely, F.F. Goldberg, N.H. Roberts, and D.F. Haasl. *Fault Tree Handbook*. US Government Printing Office, January 1981. Systems and Reliability Research, Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Commission.
2. John Viega and Gary McGraw. *Building Secure Software: How to Avoid Security Problems the Right Way*. Addison Wesley Professional, 2001.
3. Andrew P. Moore, Robert J. Ellison, and Richard C. Linger. Attack modeling for information security and survivability. Technical Report CMU/SEI-2001-TN-001, Software Engineering Institute, 2001.
4. J. D. Weiss. A system security engineering process. In *Proceedings of the 14th National Computer Security Conference*, pages 572–581, 1991.
5. Bruce Schneier. Attack trees: Modeling security threats. *Dr. Dobb's Journal*, 24(12):21–29, December 1999.
6. Kenneth S. Edge. *A Framework for Analyzing and Mitigating the Vulnerabilities of Complex Systems via Attack and Protection Trees*. PhD thesis, Air Force Institute of Technology, Ohio, 2007.
7. Jeanne H. Espedahlen. Attack trees describing security in distributed internet-enabled metrology. Master's thesis, Department of Computer Science and Media Technology, Gjøvik University College, 2007.
8. Sjouke Mauw and Martijn Oostdijk. Foundations of attack trees. In Dongho Won and Seungjoo Kim, editors, *International Conference on Information Security and Cryptology – ICISC 2005*, volume 3935 of *LNCS*, pages 186–198. Springer, 2005.
9. Alexander Opel. Design and implementation of a support tool for attack trees. Technical report, Otto-von-Guericke University, March 2005. Internship Thesis.
10. Ahto Buldas, Peeter Laud, Jaan Priisalu, Märt Saarepera, and Jan Willemson. Rational Choice of Security Measures via Multi-Parameter Attack Trees. In *Critical Information Infrastructures Security. First International Workshop, CRITIS 2006*, volume 4347 of *LNCS*, pages 235–248. Springer, 2006.
11. Ahto Buldas and Triinu Mägi. Practical security analysis of e-voting systems. In A. Miyaji, H. Kikuchi, and K. Rannenberg, editors, *Advances in Information and Computer Security, Second International Workshop on Security, IWSEC*, volume 4752 of *LNCS*, pages 320–335. Springer, 2007.
12. Aivo Jürgenson and Jan Willemson. Processing multi-parameter attacktrees with estimated parameter values. In A. Miyaji, H. Kikuchi, and K. Rannenberg, editors, *Advances in Information and Computer Security, Second International Workshop on Security, IWSEC*, volume 4752 of *LNCS*, pages 308–319. Springer, 2007.
13. Lauri Rätsep. The influence and measurability of the parameters of the security analysis of the Estonian e-voting system. MSc thesis, Tartu University, 2008. In Estonian.
14. Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, 1962.
15. Alexander Andrusenko. Multiparameter attack tree analysis software. BSc thesis, Tartu University, 2008. In Estonian.

Serial Model for Attack Tree Computations

Aivo Jürgenson[1,2], Jan Willemson[3]

[1] Tallinn University of Technology, Raja 15, 12618 Tallinn, Estonia
`aivo.jurgenson@eesti.ee`
[2] Elion Enterprises Ltd, Endla 16, 15033 Tallinn, Estonia
[3] Cybernetica, Aleksandri 8a, Tartu 51004, Estonia
`jan.willemson@gmail.com`

Abstract. In this paper we extend the standard attack tree model by introducing temporal order to the attacker's decision making process. This will allow us to model the attacker's behaviour more accurately, since this way it is possible to study his actions related to dropping some of the elementary attacks due to them becoming obsolete based on the previous success/failure results. We propose an efficient algorithm for computing the attacker's expected outcome based on the given order of the elementary attacks and discuss the pros and cons of considering general rooted directed acyclic graphs instead of plain trees as the foundations for attack modelling.

1 Introduction

Attack tree (also called threat tree) approach to security evaluation is several decades old. It has been used for tasks like fault assessment of critical systems [1] or software vulnerability analysis [2, 3]. The approach was first applied in the context of information systems (so-called *threat logic trees*) by Weiss [4] and later more widely adapted to information security by Bruce Schneier [5]. We refer to [6, 7] for good overviews on the development and applications of the methodology.

Since their first introduction, attack trees have been used to describe attacks against various real-world applications like Border Gateway Protocol [8], SCADA protocols [9] and e-voting infrastructures [10]. Attack trees have found their place in computer science education [11] and several support tools like AttackTree+[4] and SecurITree[5] have been developed.

Early approaches to attack tree modelling were mostly concerned with just categorising the attacks [8] or modelling the attacker's behaviour by one specific parameter of the attacks like the cost, difficulty or severity [5,

[4] `http://www.isograph-software.com/atpover.htm`
[5] `http://www.amenaza.com/`

9, 12]. A substantial step forward was taken by Buldas *et al.* [13] who introduced the idea of game-theoretic modelling of the attacker's decision making process based on several interconnected parameters like the cost, risks and penalties associated with different elementary attacks. This approach was later refined by Jürgenson and Willemson [14, 15] and applied to the analysis of the security of several e-voting solutions by Buldas and Mägi [10].

So far, practically all the research in the field of attack trees has concentrated on what one could call a parallel model [4, 5, 3, 8, 9, 16, 12–14, 10, 15]. Essentially, the model assumes that all the elementary attacks take place simultaneously and hence the attacker's possible decisions based on success or failure of some of the elementary attacks are ignored. However, as noted already in [15], this model is unrealistic. In practice, the attacker is able to order his actions and try different alternative scenarios if some others fail or to stop trying altogether if some critical subset of elementary attacks has already failed or succeeded. Not risking with the hopeless or unnecessary attempts clearly reduces the amount of potential penalties and hence increases the attacker's expected outcome.

The main contribution of this paper is to surpass this shortcoming by introducing what one could call a serial model for attack trees. We extend the basic parallel model with temporal order of the elementary attacks and give the attacker some flexibility in skipping some of them or stopping the attack before all of the elementary attacks have been tried. The other contribution is a generalisation of the attack tree approach to accommodate arbitrary rooted directed acyclic graphs, which will enable us to conveniently ensure consistency of our computations in the general framework proposed by Mauw and Oostdijk [12].

The paper is organised as follows. In Section 2 we first briefly review the basic multi-parameter attack tree model. Sections 3 and 4 extend it by introducing attack descriptions based on general Boolean functions and temporal order of elementary attacks, respectively. Section 5 presents an efficient algorithm for computing the attacker's expected outcome of the attack tree with the predefined order of leaves. Finally, Section 6 draws some conclusions and sets directions for further work.

2 The Attack Tree Model

Basic idea of the attack tree approach is simple – the analysis begins by identifying one *primary threat* and continues by dividing the threat into subattacks, either all or some of them being necessary to materialise the primary threat. The subattacks can be divided further etc., until we reach the state where it does not make sense to divide the resulting attacks any more; these kinds of non-splittable attacks are called *elementary attacks* and the security analyst will have to evaluate them somehow. During the splitting process, a tree is formed having the primary threat in its root and elementary attacks in its leaves. Using the structure of the tree and the estimations of the leaves, it is then (hopefully) possible to give some estimations of the root node as well. In practice, it mostly turns out to be sufficient to consider only two kinds of splits in the internal nodes of the tree, giving rise to AND- and OR-nodes. As a result, an AND-OR-tree is obtained, forming the basis of the subsequent analysis.

The crucial contribution of Buldas *et al.* [13] was the introduction of four game-theoretically motivated parameters for each leaf node of the tree. This approach was later optimised in [15], where the authors concluded that only two parameters suffice. Following their approach, we consider the set of elementary attacks $\mathcal{X} = \{X_1, X_2, \ldots, X_n\}$ and give each one of them two parameters:

- p_i – success probability of the attack X_i,
- $\mathsf{Expenses}_i$ – expected expenses (i.e. costs plus expected penalties) of the attack X_i.

Besides these parameters, there is a global value Gains expressing the benefit of the attacker if he is able to materialise the primary threat.

In the parallel model of [15], the expected outcome of the attacker is computed by maximising the expression

$$\mathsf{Outcome}_S = p_S \cdot \mathsf{Gains} - \sum_{X_i \in S} \mathsf{Expenses}_i \qquad (1)$$

over all the assignments $S \subseteq \mathcal{X}$ that make the Boolean formula $\mathcal{F}$, represented by the attack tree, true. (Here p_S denotes the success probability of the primary threat.) Like in the original model of Buldas *et al.* [13], we assume that the attacker behaves rationally, i.e. he attacks only if there is an attack scenario with a positive outcome. The defender's task is thus achieving a situation where all the attack scenarios would be non-beneficial for the attacker.

Our aim is to develop this model in two directions. In Section 3 we will generalise the attack tree model a bit to allow greater flexibility and expressive power of our model, and in Section 4 we will study the effects of introducing linear (temporal) order to the set of elementary attacks.

3 Attack Descriptions as Monotone Boolean Functions

Before proceeding, we briefly discuss a somewhat different perspective on attack tree construction. Contrary to the standard top-down ideology popularised by Schneier [5], a bottom-up approach is also possible. Say, our attacker has identified the set of elementary attacks $\mathcal{X}$ available to him and he needs to figure out, which subsets of $\mathcal{X}$ are sufficient to mount the root attack. In this paper we assume that the set of such subsets is monotone, i.e. if some set of elementary attacks suffices, then so does any of its supersets. This way it is very convenient to describe all the successful attacks by a monotone Boolean function $\mathcal{F}$ on the set of variables $\mathcal{X}$.

Of course, if we have constructed an attack tree then it naturally corresponds to a Boolean function. Unfortunately, considering only the formulae that have a tree structure is not always enough. Most notably, trees can not handle the situation, where the same lower-level attack is useful in several, otherwise independent higher-level attacks, and this is clearly a situation we can not ignore in practical security analysis.

Another shortcoming of the plain attack tree model follows from the general framework by Mauw and Oostdijk [12]. They argue that the semantics of an attack tree is inherently consistent if and only if the tree can be transformed into an equivalent form without changing the value of the expected outcome. When stating and proving their result, they essentially transform the underlying Boolean formula into a disjunctive normal form, but when doing so, they need to introduce several copies of some attacks, therefore breaking the tree structure in favour of a general rooted directed acyclic graph (RDAG). Since AND-OR-RDAGs are equivalent to monotone Boolean functions, there is no immediate need to take the generalisation any further.

Thus it would be more consistent and fruitful not to talk about *attack trees*, but rather *attack RDAGs*. On the other hand, as the structure of a tree is so much more convenient to analyse than a general RDAG, we should still try to stick to the trees whenever possible. We will see one specific example of a very efficient tree analysis algorithm in Section 5.

4 Ordering Elementary Attacks

After the attacker has selected the set of possible elementary attacks $\mathcal{X}$ and described the possible successful scenarios by means of a monotone Boolean function $\mathcal{F}$, he can start planning the attacks. Unlike the naïve parallel model of Schneier [5], the attacker has a lot of flexibility and choice. He may try some elementary attack first and based on its success or failure select the next elementary attack arbitrarily or even decide to stop attacking altogether (e.g. due to certain success or failure of the primary threat). Such a fully adaptive model is still too complicated to analyse with the current methods, thus we will limit the model to be semi-adaptive. I.e., we let the attacker to fix linear order of some elementary attacks in advance and assume that he tries them in succession, possibly skipping superfluous elementary attacks and stopping only if he knows that the Boolean value of $\mathcal{F}$ has been completely determined by the previous successes and failures of elementary attacks.

The full strategy of the attacker will be the following.

1. Create an attack RDAG with the set of leaf nodes $\mathcal{X} = \{X_1, X_2, \dots, X_n\}$.
2. Select a subset $S \subseteq \mathcal{X}$ materialising the primary threat and consider the corresponding subtree.
3. Select a permutation α of S.
4. Based on the subtree and permutation α, compute the expected outcome.
5. Maximise the expected outcome over all the choices of S and α.

This paper is mostly concerned with item 4 in the above list, but doing so we must remember that when building a complete attack analysis tool, other items can not be disregarded either. Optimisations are possible, e.g. due to monotonicity there is no need to consider any subsets of attack suites that do not materialise the primary threat. Even more can be done along the lines of [15], Section 4.1, but these aspects remain outside of the scope of the current paper.

Since only one subset S and the corresponding subtree are relevant in the above step 4, we can w.l.o.g. assume that $S = \mathcal{X}$. The attacker's behaviour for permutation α will be modelled as shown in Algorithm 1.

Consider the example attack tree depicted in Figure 1, where we assume $\alpha = id$ for better readability.

The attacker starts off by trying the elementary attack X_1. Independent of whether it succeeds or fails, there are still other components needed to complete the root attack, so he tries X_2 as well. If it fails, we see that the whole tree fails, so it does not make sense to try X_3 and X_4. If both X_1 and X_2 have succeeded, we see that it is not necessary to try X_3, since X_1 and X_3 have a common OR-parent, so success or failure of X_4 determines the final outcome. If X_1 fails and X_2 succeeds, we need the success of both X_3 and X_4 to complete the task; if one of them fails, we stop and accept the failure.

Algorithm 1 Perform the attack

```
Require: The set of elementary attacks $\mathcal{X} = \{X_1, X_2, \ldots, X_n\}$, permutation $\alpha \in S_n$
    and a monotone Boolean formula $\mathcal{F}$ describing the attack scenarios
 1: for $i := 1$ to $n$ do
 2:    Consider $X_{\alpha(i)}$
 3:    if success or failure of $X_{\alpha(i)}$ has no effect on the success or failure of the root
       node then
 4:       Skip $X_{\alpha(i)}$
 5:    else
 6:       Try to perform $X_{\alpha(i)}$
 7:       if the root node succeeds or fails then
 8:          Stop
 9:       end if
10:    end if
11: end for
```

The expected outcome of the attack based on permutation α will be defined as

$$\mathsf{Outcome}_\alpha = p_\alpha \cdot \mathsf{Gains} - \sum_{X_i \in \mathcal{X}} p_{\alpha,i} \cdot \mathsf{Expenses}_i\,, \tag{2}$$

where p_α is the success probability of the primary threat and $p_{\alpha,i}$ denotes the probability that the node X_i is encountered during Algorithm 1. Before proceeding, we will prove that the expected outcome of Algorithm 1 does not depend on the specific form of the formula $\mathcal{F}$. This essentially gives us the compliance of our attack tree model in the framework of Mauw and Oostdijk [12]. Formally, we will state and prove the following theorem, similar to Proposition 1 in [15].

Theorem 1. *Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be two monotone Boolean formulae such that $\mathcal{F}_1 \equiv \mathcal{F}_2$, and let $\mathsf{Outcome}^1_\alpha$ and $\mathsf{Outcome}^2_\alpha$ be the expected outcomes obtained running Algorithm 1 on the corresponding formulae. Then*

$$\mathsf{Outcome}^1_\alpha = \mathsf{Outcome}^2_\alpha\,.$$

Proof. We can observe that Algorithm 1 really does not depend on the attack description having a tree structure, all the decisions to skip or stop can be taken based on the Boolean function $\mathcal{F}$. Assume we have already fixed the results of the elementary attacks $X_{\alpha(1)}, \ldots, X_{\alpha(i-1)}$. Then we see that

– the node $X_{\alpha(i)}$ may be skipped if for all the values of $X_{\alpha(i+1)}, \ldots, X_{\alpha(n)}$ we have

$$\mathcal{F}\left(X_{\alpha(1)}, \ldots, X_{\alpha(i-1)}, t, X_{\alpha(i+1)}, \ldots, X_{\alpha(n)}\right) =$$

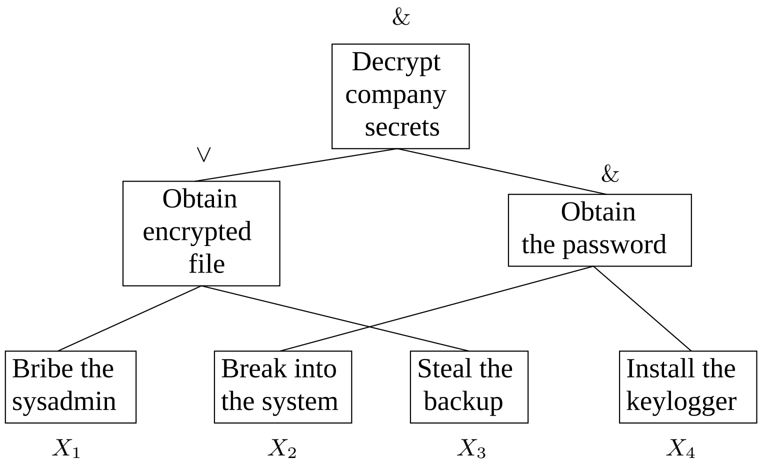


Fig. 1. An example attack tree. The left-to-right ordering of the leaf nodes in the tree represents the permutation $\alpha = id$ of the set $\mathcal{X} = \{X_1, X_2, X_3, X_4\}$.

$$= \mathcal{F}\left(X_{\alpha(1)}, \dots, X_{\alpha(i-1)}, f, X_{\alpha(i+1)}, \dots, X_{\alpha(n)}\right) ,$$

– there is no need to proceed with Algorithm 1 after the node $X_{\alpha(i)}$ if for all the values of $X_{\alpha(i+1)}, \dots, X_{\alpha(n)}$ we have

$$\mathcal{F}\left(X_{\alpha(1)}, \dots, X_{\alpha(i-1)}, X_{\alpha(i)}, X_{\alpha(i+1)}, \dots, X_{\alpha(n)}\right) = t$$

or

$$\mathcal{F}\left(X_{\alpha(1)}, \dots, X_{\alpha(i-1)}, X_{\alpha(i)}, X_{\alpha(i+1)}, \dots, X_{\alpha(n)}\right) = f .$$

□

Thus, our serial model for attack trees follows the guidelines given in Section 3 and it really is safe to talk about Boolean functions describing the attack scenarios.

Next we will show formally that introducing order to the elementary attacks really increases the attacker's expected outcome. Comparing (2) to (1) we get the following theorem.

Theorem 2. *Let $\mathcal{F}$ be a monotone Boolean function on $n \geq 2$ variables describing the attack scenarios. Let $\mathsf{Outcome}_\alpha$ be defined by (2) and let $\mathsf{Outcome}_\mathcal{X}$ be defined by (1) for $S = \mathcal{X}$. Then we have*

$$\mathsf{Outcome}_\alpha \geq \mathsf{Outcome}_\mathcal{X} . \qquad (3)$$

If for all the elementary attacks X_i ($i = 1, \dots, n$) one also has $\mathsf{Expenses}_i > 0$, then strict inequality holds in (3).

Proof. First we note that by [15] we can compute the success probability of the attacker as follows:

$$p_{\mathcal{X}} = \sum_{\substack{S \subseteq \mathcal{X} \\ \mathcal{F}(S := \mathsf{true}) = \mathsf{true}}} \prod_{X_i \in S} p_i \prod_{X_j \in \mathcal{X} \setminus S} (1 - p_j)\,,$$

where $\mathcal{F}(S := \mathsf{true})$ denotes evaluation of the Boolean function $\mathcal{F}$, when all the variables of S are assigned the value true and all others the value false. This is exactly the total probability of all the successful branches of Algorithm 1 and thus $p_{\mathcal{X}} = p_\alpha$ (implying that p_α is actually independent of α). We also have that $\forall i\, p_{\alpha,i} \leq 1$ and hence the inequality (3) follows.

Assume now that for all X_i we have $\mathsf{Expenses}_i > 0$. Then in order to prove that strict inequality holds in (3), we need to show that there exists such an index i that $p_{\alpha,i} < 1$. Consider the elementary attack $X_{\alpha(n)}$ that the attacker is supposed to try last. If there exists an evaluation of the Boolean variables $X_{\alpha(1)}, \ldots, X_{\alpha(n-1)}$ such that

$$\mathcal{F}\left(X_{\alpha(1)}, \ldots, X_{\alpha(n-1)}, t\right) = \mathcal{F}\left(X_{\alpha(1)}, \ldots, X_{\alpha(n-1)}, f\right)\,,$$

then $X_{\alpha(n)}$ is superfluous in this scenario and hence $p_{\alpha,n} < 1$.

If on the other hand we have

$$\mathcal{F}\left(X_{\alpha(1)}, \ldots, X_{\alpha(n-1)}, t\right) \neq \mathcal{F}\left(X_{\alpha(1)}, \ldots, X_{\alpha(n-1)}, f\right)$$

for all evaluations of $X_{\alpha(1)}, \ldots, X_{\alpha(n-1)}$, then due to monotonicity of $\mathcal{F}$ we can only have that

$$\mathcal{F}\left(X_{\alpha(1)}, \ldots, X_{\alpha(n-1)}, f\right) = f$$

and

$$\mathcal{F}\left(X_{\alpha(1)}, \ldots, X_{\alpha(n-1)}, t\right) = t\,,$$

implying $\mathcal{F}(Y_1, \ldots, Y_n) \equiv Y_n$. But in this case all the elementary attacks before the last one get skipped, so $p_{\alpha,1} = \ldots = p_{\alpha,n-1} = 0$. □

Thus, introducing ordering of the elementary attacks is guaranteed to give at least as good a result to the attacker as the routine described in [15]. In the interesting case, when all attack components have positive expenses, the attacker's expected outcome is strictly larger.

5 Computing the Expected Outcome

There are $n+1$ parameters that need to be computed in order to find the expected outcome using the formula (2) – the total success probability p_α and the probabilities $p_{\alpha,i}$ that the node X_i is encountered during Algorithm 1. It turns out that there is an efficient algorithm for computing these quantities provided that the given monotone Boolean function can actually be described by a tree. In what follows we will also assume that the tree is binary, but this restriction is not a crucial one.

So let us have an attack tree with the leaf nodes $X_1, \ldots, X_n$ and the corresponding success probabilities p_i, $i = 1, \ldots, n$. We will assume that all these probabilities are independent and consider the permutation $\alpha \in S_n$. In order to explain the algorithm, we first introduce three extra parameters to each node Y, namely $Y.t$, $Y.f$ and $Y.u$ showing the probabilities that the node has been proven to be respectively true, false or yet undefined in the course of the analysis. Initially, we may set $Y.t = Y.f = 0$ and $Y.u = 1$ for all the nodes and the algorithm will work by incrementally adjusting these values, so that in the end of the process we will have $R.t = p_\alpha$ for the root node R. Throughout the computations we will of course retain the invariant $Y.t + Y.f + Y.u = 1$ for all the nodes Y, hence one of these parameters is actually superfluous. In the presentation version of the algorithm we will drop the parameter $Y.u$, even though it actually plays the central role.

Going back to the high-level description of Algorithm 1, we see that the most difficult step is step 3, where the attacker is supposed to find out whether the next elementary attack in his list may have any effect on the success or failure of the root node. Elementary attack does not have any effect iff there is a node on the path from that particular leaf to the root that has already been proven to be true or false. Thus the next elementary attack should be tried iff all the nodes on this path are undefined – and this is precisely the event that gives us the required probability $p_{\alpha,i}$.

Let the path from root R to the leaf X_i then be $(Y_0 = R, Y_1, \ldots, Y_m = X_i)$. Thus, we need to compute the probability

$$\begin{aligned} p_{\alpha,i} &= \Pr[Y_0 = u \,\&\, Y_1 = u \,\&\, \ldots \,\&\, Y_m = u] = \\ &= \Pr[Y_0 = u \,|\, Y_1 = u\,, \ldots\,, Y_m = u] \cdot \\ &\quad \cdot \Pr[Y_1 = u \,|\, Y_2 = u\,, \ldots\,, Y_m = u] \cdot \ldots \\ &\quad \ldots \cdot \Pr[Y_{m-1} = u \,|\, Y_m = u] \cdot \Pr[Y_m = u] = \\ &= \Pr[Y_0 = u \,|\, Y_1 = u] \cdot \Pr[Y_1 = u \,|\, Y_2 = u] \cdot \ldots \\ &\quad \ldots \cdot \Pr[Y_{m-1} = u \,|\, Y_m = u] \cdot \Pr[Y_m = u] \end{aligned} \tag{4}$$

The equations

$$\Pr[Y_k = u \,|\, Y_{k+1} = u\,,\ \ldots\,,\ Y_m = u] = \Pr[Y_k = u \,|\, Y_{k+1} = u]$$

hold due to the tree structure of our underlying RDAG and the independence assumption of the elementary attacks. In (4) we have $\Pr[Y_m = u] = \Pr[X_i = u] = 1$ and all the other probabilities are of the form $\Pr[Y_k = u \,|\, Y_{k+1} = u]$. Hence, we need to evaluate the probability that the parent node Y_k is undefined assuming that one of its children, Y_{k+1}, is undefined. This probability now depends on whether Y_k is an AND- or OR-node. If Y_k is an AND-node and Y_{k+1} is undefined, then so is Y_k, if its other child Z is either true or undefined, which is the case with probability $Z.t + Z.u = 1 - Z.f$. Similarly, if Y_k is an OR-node and Y_{k+1} is undefined, then so is Y_k, if its other child Z is either false or undefined, which is the case with probability $Z.f + Z.u = 1 - Z.t$.

This way, (4) gives an efficient way of computing $p_{\alpha,i}$ assuming that the current parameters of the internal nodes of the tree are known. Hence, we need the routines to update these as well. These routines are straightforward. If the elementary attack X_i is tried, only the parameters of the nodes on the path $(Y_m = X_i, \ldots, Y_1, Y_0 = R)$ from that leaf to the root need to be changed. We do it by first setting $Y_m.t = p_i$, $Y_m.f = 1 - p_i$ and $Y_m.u = 0$ and then proceed towards the root. If the node we encounter is AND-node A with children B and C, we set

$$A.t = B.t \cdot C.t\,, \tag{5}$$

$$A.f = B.f + C.f - B.f \cdot C.f\,, \tag{6}$$

and if we encounter an OR-node A with children B and C, we set

$$A.t = B.t + C.t - B.t \cdot C.t\,, \tag{7}$$

$$A.f = B.f \cdot C.f\,. \tag{8}$$

As noted above, we see that the quantities $Y.u$ are actually never needed in the computations.

This way we get the full routine described as Algorithm 2.

Algorithm 2 is very efficient. In order to compute the $n + 1$ necessary probabilities, it makes one run through all the leaves of the tree and at each run the path from the leaf to the root is traversed twice. Since the number of vertices on such a path in a (binary) tree can not be larger than the number of leaves n, we get that the worst-case time complexity of Algorithm 2 is $O(n^2)$. If the tree is roughly balanced, this estimate

Algorithm 2 Computing the probabilities $p_{\alpha,i}$

```
Require: An attack tree with leaf set $\mathcal{X} = \{X_1, X_2, \dots, X_n\}$ and a permutation $\alpha \in S_n$
Ensure: The probabilities $p_{\alpha,i}$ for $i = 1, 2, \dots, n$
 1: for all $Z \in \{X_1, \dots, X_n\}$ do
 2:     $Z.t := 0,\ Z.f := 0$
 3: end for
 4: for $i := 1$ to $n$ do
 5:     Find the path $(Y_0, Y_1, \dots, Y_m)$ from the root $Y_0 = R$ to the leaf $Y_m = X_{\alpha(i)}$
 6:     $p_{\alpha,\alpha(i)} := \prod_{j=1}^{m}(1 - Z_j.a)$, where $Z_j$ is the sibling node of $Y_j$ and
            $a = \begin{cases} t, & \text{if } Y_{j-1} \text{ is an OR-node,} \\ f, & \text{if } Y_{j-1} \text{ is an AND-node} \end{cases}$
 7:     $X_{\alpha(i)}.t = p_{\alpha(i)}$
 8:     $X_{\alpha(i)}.f = 1 - p_{\alpha(i)}$
 9:     Update the parameters of the nodes $Y_{m-1}, Y_{m-2}, \dots, Y_0$ according to formulae (5)–(8)
10: end for
```

drops even to $O(n \log n)$. This is a huge performance increase compared to a naïve algorithm that one could design based on the complete attack scenario analysis described after Figure 1 in Section 4. We studied the naïve algorithm and it turns out that it is not only worst-case exponential, but also average-case exponential [17].

Of course, as noted in Section 4, Algorithm 2 is only one building block in the whole attack tree analysis. In order to find out the best attack strategy of the attacker, we should currently consider all the subsets of $\mathcal{X}$ and all their permutations. Optimisation results presented in [14] give a strong indication that a vast majority of the possible cases can actually be pruned out, but these methods remain outside of the scope of the current paper.

6 Conclusions and Further Work

In this paper we studied the effect of introducing a temporal order of elementary attacks into the attacker's decision making process together with some flexibility in retreating of some of them. It turns out that taking temporal dependencies into account allows the attacker to achieve better expected outcomes and as such, it brings the attack tree model one step closer to the reality. This reality comes for a price of immense increase in computational complexity, if we want to compute the attacker's exact outcome by considering all the possible scenarios in a naïve way.

Thus there are two main challenges for the future research. First, one may try to come up with optimisations to the computational process and in this paper we showed one possible optimisation which works well for attack trees. The second approach is approximation. In attack tree analysis we are usually not that much interested in the exact maximal outcome of the attacker, but we rather want to know whether it is positive or negative. This observation gives us huge potential for rough estimates, which still need to be studied, implemented and tried out in practice.

In this paper we limited ourselves to a semi-adaptive model, where the attacker is bound to the predefined order of elementary attacks and may only choose to drop some of them. Fully adaptive case where the attacker may choose the next elementary attack freely is of course even more realistic, but it is currently too complicated to analyse. Our model is also non-blocking in the sense that there are no elementary attacks, failure of which would block execution of the whole tree. However, in practice it happens that when failing some attack, the attacker might get jailed and is unable to carry on. Hence, future studies in the area of adaptive and possibly-blocking case are necessary.

As a little technical contribution we also discussed the somewhat inevitable generalisation of attack trees to RDAGs, but our results also show that whenever possible, we should still stick to the tree structure. Possible optimisations of RDAG-based algorithms remain the subject for future research as well.

7 Acknowledgments

This research was supported by Estonian Science Foundation grant no 7081. The authors are grateful to Margus Niitsoo for his discussions and helpful comments.

References

1. Vesely, W., Goldberg, F., Roberts, N., Haasl, D.: Fault Tree Handbook. US Government Printing Office (January 1981) Systems and Reliability Research, Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Commission.
2. Viega, J., McGraw, G.: Building Secure Software: How to Avoid Security Problems the Right Way. Addison Wesley Professional (2001)
3. Moore, A.P., Ellison, R.J., Linger, R.C.: Attack modeling for information security and survivability. Technical Report CMU/SEI-2001-TN-001, Software Engineering Institute (2001)
4. Weiss, J.D.: A system security engineering process. In: Proceedings of the 14th National Computer Security Conference. (1991) 572–581

5. Schneier, B.: Attack trees: Modeling security threats. Dr. Dobb's Journal **24**(12) (December 1999) 21–29
6. Edge, K.S.: A Framework for Analyzing and Mitigating the Vulnerabilities of Complex Systems via Attack and Protection Trees. PhD thesis, Air Force Institute of Technology, Ohio (2007)
7. Espedahlen, J.H.: Attack trees describing security in distributed internet-enabled metrology. Master's thesis, Department of Computer Science and Media Technology, Gjøvik University College (2007)
8. Convery, S., Cook, D., Franz, M.: An attack tree for the border gateway protocol. IETF Internet draft (Feb 2004) Available at `http://www.ietf.org/proceedings/04aug/I-D/draft-ietf-rpsec-bgpattack-00.txt`.
9. Byres, E., Franz, M., Miller, D.: The use of attack trees in assessing vulnerabilities in SCADA systems. In: International Infrastructure Survivability Workshop (IISW'04), IEEE, Lisbon, Portugal. (2004)
10. Buldas, A., Mägi, T.: Practical security analysis of e-voting systems. In Miyaji, A., Kikuchi, H., Rannenberg, K., eds.: Advances in Information and Computer Security, Second International Workshop on Security, IWSEC. Volume 4752 of LNCS., Springer (2007) 320–335
11. Saini, V., Duan, Q., Paruchuri, V.: Threat modeling using attack trees. J. Comput. Small Coll. **23**(4) (2008) 124–131
12. Mauw, S., Oostdijk, M.: Foundations of attack trees. In Won, D., Kim, S., eds.: International Conference on Information Security and Cryptology – ICISC 2005. Volume 3935 of LNCS., Springer (2005) 186–198
13. Buldas, A., Laud, P., Priisalu, J., Saarepera, M., Willemson, J.: Rational Choice of Security Measures via Multi-Parameter Attack Trees. In: Critical Information Infrastructures Security. First International Workshop, CRITIS 2006. Volume 4347 of LNCS., Springer (2006) 235–248
14. Jürgenson, A., Willemson, J.: Processing multi-parameter attacktrees with estimated parameter values. In Miyaji, A., Kikuchi, H., Rannenberg, K., eds.: Advances in Information and Computer Security, Second International Workshop on Security, IWSEC. Volume 4752 of LNCS., Springer (2007) 308–319
15. Jürgenson, A., Willemson, J.: Computing exact outcomes of multi-parameter attack trees. In: On the Move to Meaningful Internet Systems: OTM 2008. Volume 5332 of LNCS., Springer (2008) 1036–1051
16. Opel, A.: Design and implementation of a support tool for attack trees. Technical report, Otto-von-Guericke University (March 2005) Internship Thesis.
17. Jürgenson, A., Willemson, J.: Ründepuud: pooladaptiivne mudel ja ligikaudsed arvutused (in Estonian). Technical Report T-4-4, Cybernetica, Institute of Information Security (2009) `http://research.cyber.ee/`.

On Fast and Approximate Attack Tree Computations

Aivo Jürgenson[1,2], Jan Willemson[3]

[1] Tallinn University of Technology, Raja 15,Tallinn 12618, Estonia
aivo.jurgenson@eesti.ee
[2] Cybernetica, Akadeemia 21, Tallinn 12618, Estonia
[3] Cybernetica, Aleksandri 8a, Tartu 51004, Estonia
jan.willemson@gmail.com

Abstract. In this paper we address the problem of inefficiency of exact attack tree computations. We propose several implementation-level optimizations and introduce a genetic algorithm for fast approximate computations. Our experiments show that for attack trees having less than 30 leaves, the confidence level of 89% can be achieved within 2 seconds using this algorithm. The approximation scales very well and attack trees of practical size (up to 100 leaves) can be analyzed within a few minutes.

1 Introduction

Structural methods for security assessment have been used for several decades already. Called *fault trees* and applied to analyse general security-critical systems in early 1980-s [1], they were adjusted for information systems and called *threat logic trees* by Weiss in 1991 [2]. In the late 1990-s, the method was popularized by Schneier under the name *attack trees* [3]. Since then, it has evolved in different directions and has been used to analyze the security of several practical applications, including PGP [4], Border Gateway Protocol [5], SCADA systems [6], e-voting systems [7], etc. We refer to [8, 9] for good overviews on the development and applications of the methodology.

Even though already Weiss [2] realized that the attack components may have several parameters in practice, early studies mostly focused on attack trees as a mere attack dependence description tool and were limited to considering at most one parameter at a time [3, 10, 11]. A substantial step forward was taken by Buldas *et al.* [12] who introduced the idea of game-theoretic modeling of the attacker's decision making process based on several interconnected parameters like the cost, risks and penalties associated with different elementary attacks. This approach was later refined by Jürgenson and Willemson by first extending the parameter domain from point values to interval estimates [13] and then by creating the first semantics for multi-parameter attack trees, consistent with the general framework of Mauw and Oostdijk [11, 14].

Even though being theoretically sound, the results of Jürgenson and Willemson are rather discouraging from an engineering point of view. Even with all the optimizations proposed in [14], they are still able to analyze the trees of at most 20 leaves in reasonable time and this may not suffice for many practical applications. Hence, the aim of this paper is to improve their results in two directions. First, we implement several additional optimizations and second, we create and test a genetic algorithm for fast approximations.

The paper is organized as follows. First, in Section 2 we will briefly define attack trees and the required parameters. Then Section 3 will explain our new set of optimizations, which in turn will be tested for performance in Section 4. Section 5 will cover our genetic algorithm and finally Section 6 will draw some conclusions.

2 Attack Trees

Basic idea of the attack tree approach is simple – the analysis begins by identifying one *primary threat* and continues by dividing the threat into subattacks, either all or some of them being necessary to materialize the primary threat. The subattacks can be divided further etc., until we reach the state where it does not make sense to divide the resulting attacks any more; these kinds of non-splittable attacks are called *elementary attacks* and the security analyst will have to evaluate them somehow.

During the splitting process, a tree is formed having the primary threat in its root and elementary attacks in its leaves. Using the structure of the tree and the estimations of the leaves, it is then (hopefully) possible to give some estimations of the root node as well. In practice, it mostly turns out to be sufficient to consider only two kinds of splits in the internal nodes of the tree, giving rise to AND- and OR-nodes. As a result, an AND-OR-tree is obtained, forming the basis of the subsequent analysis. We will later identify the tree as a (monotone) Boolean formula built on the set of elementary attacks as literals.

The crucial contribution of Buldas *et al.* [12] was the introduction of four game-theoretically motivated parameters for each leaf node of the tree. This approach was later optimized in [14], where the authors concluded that only two parameters suffice. Following their approach, we consider the set of elementary attacks $\mathcal{X} = \{X_1, X_2, \ldots, X_n\}$ and give each one of them two parameters:

- p_i – success probability of the attack X_i,
- $\mathsf{Expenses}_i$ – expected expenses (i.e. costs plus expected penalties) of the attack X_i.

Besides these parameters, there is a global value Gains expressing the benefit of the attacker if he is able to materialize the primary threat.

3 Efficient Attack Tree Computations

Let us have the attack tree expressed by the monotone Boolean formula $\mathcal{F}$ built on the set of elementary attacks $\mathcal{X} = \{X_1, X_2, \ldots, X_n\}$. In the model of [14], the expected outcome of the attacker is computed by maximizing the expression

$$\mathsf{Outcome}_\sigma = p_\sigma \cdot \mathsf{Gains} - \sum_{X_i \in \sigma} \mathsf{Expenses}_i \tag{1}$$

over all the assignments $\sigma \subseteq \mathcal{X}$ that make the Boolean formula $\mathcal{F}$ true. Here p_σ denotes the success probability of the primary threat and as shown in [14], this quantity can be computed in time linear in the number n of elementary attacks. The real complexity of maximizing (1) comes from the need to go through potentially all the 2^n subsets $\sigma \subseteq \mathcal{X}$. Of course, there are some useful observations to make.

- The Boolean function $\mathcal{F}$ is monotone and we are only interested in the satisfying assignments σ. Hence, it is not necessary to consider subsets of non-satisfying assignments.
- In [14], Theorem 1, it was proven that if for some AND-node in the attack tree the assignment σ evaluates some of its children as true and others as false, this σ can be disregarded without affecting the correct outcome.

We start the contributions of the current paper by additionally noting that the DPLL algorithm [15], used in [14] to generate all the satisfying assignments, introduces a lot of unnecessary overhead. The formula $\mathcal{F}$ first needs to be transformed to CNF and later maintained as a set of clauses, which, in turn, are sets of literals. Since set is a very inconvenient data structure to handle in the computer, we can hope for some performance increase by dropping it in favor of something more efficient.

In our new implementation, we keep the formula $\mathcal{F}$ as it is – in the form of a tree. The assignments σ are stored as sequences of ternary bits, i.e. strings of three possible values t, f and u (standing for true, false and undefined, respectively). The computation rules of the corresponding ternary logic are natural, see Table 1.

&	t	f	u
t	t	f	u
f	f	f	f
u	u	f	u

$\vee$	t	f	u
t	t	t	t
f	t	f	u
u	t	u	u

Table 1. Computation rules for ternary logic

In its core, our new algorithm still follows the approach of DPLL – we start off with the assignment $[\mathsf{u}, \mathsf{u}, \ldots, \mathsf{u}]$ and proceed by successively trying to evaluate the literals as f and t. Whenever we reach the value t for the formula $\mathcal{F}$, we know that all the remaining u-values may be arbitrary and it is not necessary to take the recursion any deeper. Similarly, when we obtain the value f for the formula $\mathcal{F}$, we know that no assignment of u-values can make $\mathcal{F}$ valid. Thus the only case where we need to continue recursively, is when we have $\mathcal{F} = \mathsf{u}$. This way we obtain Algorithm 1, triggered by `process_satisfying_assignments`$([\mathsf{u}, \mathsf{u}, \ldots, \mathsf{u}])$.

Algorithm 1 Finding the satisfying assignments

```
Require: Boolean formula $\mathcal{F}$ corresponding to the given AND-OR-tree
 1: Procedure process_satisfying_assignments($\sigma$)
 2: Evaluate $\mathcal{F}(\sigma)$
 3: if $\mathcal{F}(\sigma) = \mathsf{f}$ then
 4:     Return;
 5: end if
 6: if $\mathcal{F}(\sigma) = \mathsf{t}$ then
 7:     Output all the assignments obtained from $\sigma$ by setting all its u-values to t and
        f in all the possible ways;
        Return;
 8: end if
 9: //reaching here we know that $\mathcal{F}(\sigma) = \mathsf{u}$
10: Choose $X_i$ such that $\sigma(X_i) = \mathsf{u}$
11: process_satisfying_assignments($\sigma/[X_i := \mathsf{f}]$);
12: process_satisfying_assignments($\sigma/[X_i := \mathsf{t}]$);
```

Even though being conceptually simple, Algorithm 1 contains several hidden options for optimization. The first step to pay attention to lies already in line 2, the evaluation of $\mathcal{F}(\sigma)$. The evaluation process naturally follows the tree structure of $\mathcal{F}$, moving from the leaves to the root using the computation rules given by Table 1. However, taking into account Theorem 1 of [14] (see the second observation above), we can conclude that whenever we encounter a node requiring evaluation of t&f or f&t, we can abort this branch of the recursion immediately, since there is a global optimum outcome in some other branch.

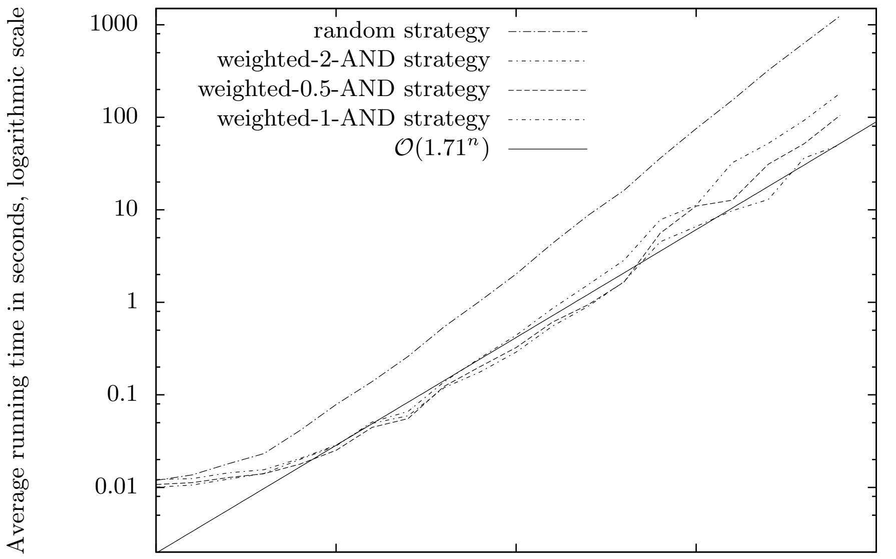
In the implementation, this kind of exception to evaluation is modelled as additional data type shortcut-false to the already existing true, false and undefined Boolean types. If the situation is encountered during the recursive evaluation of $\mathcal{F}$, shortcut-false is returned immediately to the highest level and the entire branch of processing is dropped.

Another, somewhat more obvious optimization lies within the line 10 of Algorithm 1. Of course it would be the simplest to pick the next undefined literal randomly (or in some predefined order, which gives the same result for randomly-generated test trees). However, intuitively this approach is one of the worst possible, since the working time of the algorithm depends on the number of the generated satisfying assignments. Hence, the algorithm will be faster if we can systematically disregard larger recursive branches. This is the reason why we first assign undefined literals as f on line 11 and check first, if the whole formula has become non-satisfied.

Still, a clever choice of the order of the undefined literals to specify can speed up this process even further. We implemented and tested several possible strategies.

1. Random – the next undefined literal is picked randomly.
2. Most-AND – for each undefined literal we compute the number of AND-nodes on the path from the corresponding leaf to the root and pick the onest with the highest score first.
3. Weighted-AND – the ordering routine is similar to Most-AND, but all the AND-nodes on the path do not have an equal weight. The intuition behind this approach is that when we can exclude a larger subtree, we should be able to cut off more hopeless recursion branches as well, hence it makes more sense to prefer paths with AND-nodes closer to the root. Thus we gave each node on the distance i from the root the weight $1/c^i$, where c is a predefined constant. In our experiments we used the values $c = 2$. For comparison, we also ran tests with $c = 0.5$

Fig. 1. Performance test results of different strategies for choosing undefined literals



and $c = 1$. (Not that the Most-AND strategy is equivalent to the Weighted-AND strategy with $c = 1$.)

4 Performance analysis

Somewhat surprisingly it turned out that giving more weight to the AND nodes which are closer to the root node does not necessarily help. The weighting constant $c = 0.5$ gave also very good results and in some cases better than the Weighted-2-AND strategy.

We generated random sample attack trees with 5 leaves up to 29 leaves, at least 100 trees in each group, and measured the solving time with our optimized realization and with different strategies. The results are depicted in Fig. 1.

To estimate the complexity of our algorithms, we used the least-squares method to fit a function $a^{-1} \cdot b^n$ to the running times of our best strategy method. Since there is no reasonable analytical way to establish the time complexity of our algorithm, this approach provided a quick and easy way to estimate it. The found parameters to fit the data points of the best solution (the (1)-AND method) optimally were $a = 109828$ and $b = 1.71018$. Hence we can conclude that the average complexity of our algorithm with our generated sample data is in the range of $\sim \mathcal{O}(1.71^n)$.

The average complexity estimations for all strategies were the following:

- Random strategy – $\mathcal{O}(1.90^n)$
- Weighted-2-AND strategy – $\mathcal{O}(1.78^n)$
- Weighted-1-AND strategy – $\mathcal{O}(1.71^n)$
- Weighted-0.5-AND strategy – $\mathcal{O}(1.75^n)$

However, it should be noted that differences between the Weighted-c-AND strategies were quite small and within the margin of error. Therefore, no conclusive results can be given regarding the different weighting constants. It is clear though that all the tested strategies are better than just choosing the leafs in random.

Currently the world's best #3-SAT problem solving algorithm by Konstantin Kutzkov ([16]) has the worst-case complexity $\mathcal{O}(1.6423^n)$. As #SAT problems can be parsinomically and in polynomial time converted to the #3-SAT problems (see [17], chapter 26), we can roughly compare our algorithm complexity and the #3-SAT problem solver complexity.

Direct comparison is however not possible for several reasons. First, our estimate is heuristic and is based on experimental data. Second, we are not only counting all the possible SAT solutions to the formula $\mathcal{F}$, but we actually have to generate many of them. At the same time, we are using optimizations described in Section 3.

Comparison with the result of Kutzkov still shows that our approach works roughly as well as one would expect based on the similarity of our problem setting to #SAT. It remains an open problem to develop more direct comparison methods and to find out whether some of the techniques of #SAT solvers could be adapted to the attack trees directly.

5 Approximation

Even though reimplementation in C++ and various optimization techniques described in Section 3 helped us to increase the performance of attack tree analysis significantly compared to [14], we are still practically limited to the trees having at most 30 leaves. Given the exponential nature of the problem, it is hard to expect substantial progress in the exact computations.

In order to find out, how well the exact outcome of the attack tree can be approximated, we implemented a genetic algorithm (GA) for the computations. (See [18] for an introduction into GAs.) Let us have an attack tree described by the Boolean formula $\mathcal{F}$ and the set of leaves $\mathcal{X} = \{X_1, X_2, \ldots, X_n\}$ having the parameters as described in 3. We will specify the following concepts for our GA.

Individual: any subset $\sigma \subseteq \mathcal{X}$. The individuals are internally represented by bitstrings of length n, where 1 in position i means that $X_i \in \sigma$ and 0 in position i means that $X_i \notin \sigma$.

Live individual: $\sigma \subseteq \mathcal{X}$ such that $\mathcal{F}(\sigma := \mathsf{t}) = \mathsf{t}$, i.e. such that the value of $\mathcal{F}$ is t when all the literals of σ are set to t and all the others to f.

Dead individual: $\sigma \subseteq \mathcal{X}$ such that $\mathcal{F}(\sigma := \mathsf{t}) = \mathsf{f}$.

Generation: a set of p live individuals (where p is a system-wide parameter to be determined later).

Fitness function: $\mathsf{Outcome}_\sigma$. Note that σ must be alive in order for the fitness function to be well-defined.

Crossover: in order to cross two individuals σ_1 and σ_2, we iterate throughout all the elementary attacks X_i $(i = 1, \ldots, n)$ and decide by a fair coin toss, whether we should take the descendant's ith bit from σ_1 or σ_2.

Mutation: when an individual σ needs to be mutated, a biased coin is flipped for every leaf X_i $(i = 1, \ldots, n)$ and its status (included/excluded) in σ will be changed if the coin shows heads. Since σ is internally kept as a bit sequence, mutation is accomplished by simple bit flipping.

In order to start the GA, the first generation of p live individuals must be created. We generate them randomly, using the following recursive routine.

1. Consider the root node.
2. If the node we consider is a leaf, then include it to σ and stop.
3. If the node we consider is an AND-node, consider all its descendants recursively going back to Step 2.
4. If the node we consider is an OR-node, flip a fair coin for all of them to decide whether they should be considered or not. If none of the descendants was chosen, flip the coin again for all of them. Repeat until at least one descendant gets chosen. Continue with Step 2.

It is easy to see that the resulting σ is guaranteed to be live and that the routine stops with probability 1.

Having produced our first generation of individuals $\sigma_1, \dots, \sigma_p$ (not all of them being necessarily distinct), we start the reproduction process.

1. All the individuals σ_i are crossed with everybody else, producing $\binom{p}{2}$ new individuals.
2. Each individual is mutated with probability 0.1 and for each one of them, the bias 0.1 is used.
3. The original individuals $\sigma_1, \dots, \sigma_p$ are added to the candidate (multi)set. (Note that this guarantees the existence of p live individuals.)
4. All the candidates are checked for liveness (by evaluating $\mathcal{F}(\sigma = \mathsf{t})$) and only the live ones are left.
5. Finally, p fittest individuals are selected for the next generation.

The reproduction takes place for g rounds, where g is also a system-wide parameter yet to be determined.

Next we estimate the time complexity of our GA.

– Generating p live individuals takes $\mathcal{O}(np)$ steps.
– Creating a new candidate generation by crossing takes $\mathcal{O}(np^2)$ steps.
– Mutating the candidate generation takes $\mathcal{O}(np^2)$ steps.
– Verifying liveliness takes $\mathcal{O}(np^2)$ steps.
– Computing the outcomes of live individuals takes $\mathcal{O}(np^2)$ steps.
– Sorting out the p best individuals out of the $\binom{p}{2} + p$ individuals takes $\mathcal{O}(p^2 \log p))$ steps.

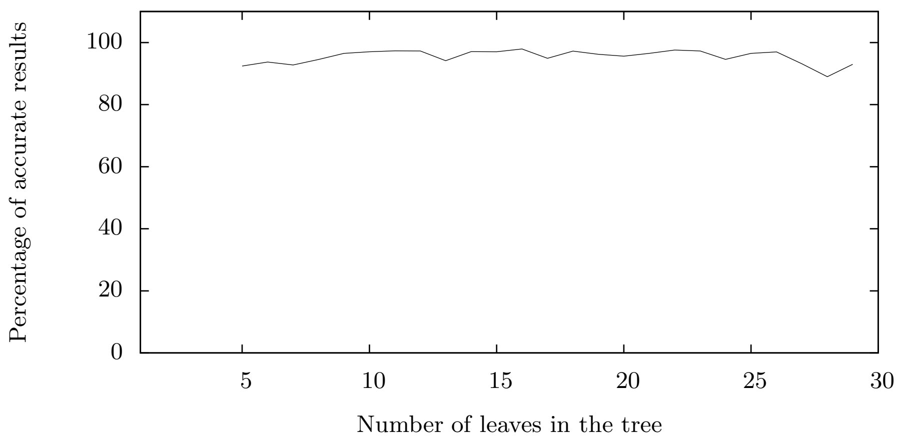
Since these steps are repeated for g generations, we can find the overall time complexity to be $\mathcal{O}(gp^2(\log p + n))$.

Of course, a GA does not guarantee that the final outcome is the best one globally. To find out, how large populations and how many iterations one has to use to hit the global maximum outcome with some degree of certainty, we performed series of tests.

First we generated a sample random set of about 6000 trees (having $n = 5 \dots 29$ leaves) and computed the exact outcome for each one of them as described in Sections 3 and 4. Next we ran our GA for population sizes $p = 5, 10, 15, \dots, 60$ and the number of iterations $1, 2, \dots, 200$. We recorded the average running times and attacker's estimated outcomes, comparing the latter ones to the exact outcomes computed before.

As a result of our tests we can say that GA allows us to reach high level of confidence (say, with 90% accuracy) very fast. There are many possible reasonable choices for $p = p(n)$ and $g = g(n)$. For example, taking $p = 2n$ and $g = 2n$ allowed us to reach 89% level of confidence for all the tree sizes of up to 29 leaves (see Fig. 2). By accuracy we here

Fig. 2. Accuracy of genetics algorithm with $p = 2n$ and $g = 2n$



mean the ratio of the trees actually computed correctly by our GA when compared to the exact outcome.

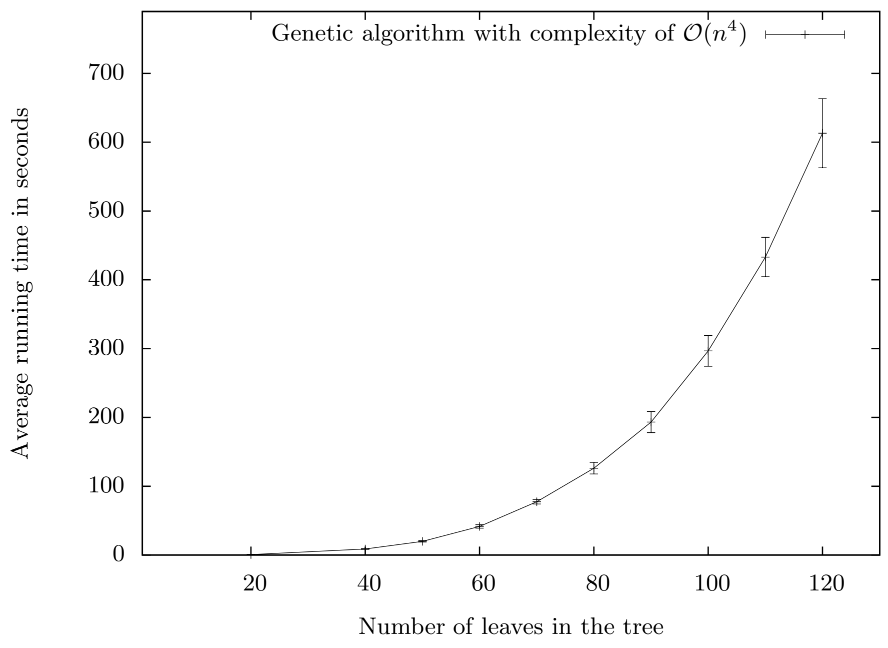
The theoretical time complexity of our GA is in this case $\mathcal{O}(n^4)$, which in reality required up to roughly 2 seconds for trees with less than 30 leaves on 2.33 GHz Intel Xeon processor. The same approach also enables us to process moderate size attack trees (70-80 leaves) in reasonable time (1-2 minutes). Attack trees of this size are helpful in analyzing real-life information systems and the multi-parameter attack trees can be now used in practical security analysis. The performance results for larger trees are given in Fig. 3. For each data point we generated 10 random trees and the average running times were measured for the genetic algorithm with parameters $p = 2n$ and $g = 2n$. The error bars represent the standard deviation of the average running time.

6 Conclusions and Further Work

In this paper we reviewed the method proposed by Jürgenson and Willemson for computing the exact outcome of a multi-parameter attack tree [14]. We proposed and implemented several optimizations and this allowed us to move the horizon of computability from the trees having 20 leaves (as in [14]) to the trees with roughly 30 leaves.

However, computing the exact outcome of an attack tree is an inherently exponential problem, hence mere optimizations on the implementa-

Fig. 3. Performance results of the genetic algorithm



tion level are rather limited. Thus we also considered an approximation technique based on genetic programming. This approach turned out to be very successful, allowing us to reach 89% of confidence within 2 seconds of computation for the trees having up to 29 leaves. The genetic algorithm is also very well scalable, making it practical to analyze even the trees having more than 100 leaves.

When running a genetic approximation algorithm, we are essentially computing a lower bound to the attacker's expected outcome. Still, an upper bound (showing that the attacker can not achieve *more* than some amount) would be much more interesting in practice. Hence, the problem of finding efficient upper bounds remains an interesting challenge for future research.

Another interesting direction is extending the model of attack tree computations. For example, Jürgenson and Willemson have also considered the serial model, where the attacker can make his decisions based on previous success or failure of elementary attacks [19]. It turns out that finding the best permutation of the elementary attacks may turn computing the optimal expected outcome into a super-exponential problem, hence the use of good approximation methods becomes inevitable in that setting.

7 Acknowledgments

This research was supported by the European Regional Development Fund through the Estonian Center of Excellence in Computer Science, EXCS, and Estonian Science Foundation grant no 8124.

References

1. Vesely, W., Goldberg, F., Roberts, N., Haasl, D.: Fault Tree Handbook. US Government Printing Office (January 1981) Systems and Reliability Research, Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Commission.
2. Weiss, J.D.: A system security engineering process. In: Proceedings of the 14th National Computer Security Conference. (1991) 572–581
3. Schneier, B.: Attack trees: Modeling security threats. Dr. Dobb's Journal **24**(12) (December 1999) 21–29
4. Schneier, B.: Secrets & Lies. Digital Security in a Networked World. John Wiley & Sons (2000)
5. Convery, S., Cook, D., Franz, M.: An attack tree for the border gateway protocol. IETF Internet draft (Feb 2004) Available at http://www.ietf.org/proceedings/04aug/I-D/draft-ietf-rpsec-bgpattack-00.txt.
6. Byres, E., Franz, M., Miller, D.: The use of attack trees in assessing vulnerabilities in SCADA systems. In: International Infrastructure Survivability Workshop (IISW'04), IEEE, Lisbon, Portugal. (2004)
7. Buldas, A., Mägi, T.: Practical security analysis of e-voting systems. In Miyaji, A., Kikuchi, H., Rannenberg, K., eds.: Advances in Information and Computer Security, Second International Workshop on Security, IWSEC. Volume 4752 of LNCS., Springer (2007) 320–335
8. Edge, K.S.: A Framework for Analyzing and Mitigating the Vulnerabilities of Complex Systems via Attack and Protection Trees. PhD thesis, Air Force Institute of Technology, Ohio (2007)
9. Espedahlen, J.H.: Attack trees describing security in distributed internet-enabled metrology. Master's thesis, Department of Computer Science and Media Technology, Gjøvik University College (2007)
10. Moore, A.P., Ellison, R.J., Linger, R.C.: Attack modeling for information security and survivability. Technical Report CMU/SEI-2001-TN-001, Software Engineering Institute (2001)
11. Mauw, S., Oostdijk, M.: Foundations of attack trees. In Won, D., Kim, S., eds.: International Conference on Information Security and Cryptology – ICISC 2005. Volume 3935 of LNCS., Springer (2005) 186–198
12. Buldas, A., Laud, P., Priisalu, J., Saarepera, M., Willemson, J.: Rational Choice of Security Measures via Multi-Parameter Attack Trees. In: Critical Information Infrastructures Security. First International Workshop, CRITIS 2006. Volume 4347 of LNCS., Springer (2006) 235–248

13. Jürgenson, A., Willemson, J.: Processing multi-parameter attacktrees with estimated parameter values. In Miyaji, A., Kikuchi, H., Rannenberg, K., eds.: Advances in Information and Computer Security, Second International Workshop on Security, IWSEC. Volume 4752 of LNCS., Springer (2007) 308–319
14. Jürgenson, A., Willemson, J.: Computing exact outcomes of multi-parameter attack trees. In: On the Move to Meaningful Internet Systems: OTM 2008. Volume 5332 of LNCS., Springer (2008) 1036–1051
15. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem proving. Communications of the ACM **5**(7) (1962) 394–397
16. Kutzkov, K.: New upper bound for the #3-sat problem. Inf. Process. Lett. **105**(1) (2007) 1–5
17. Kozen, D.: The design and analysis of algorithms. Springer (1992)
18. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1989)
19. Jürgenson, A., Willemson, J.: Serial model for attack tree computations. In: Proceedings of ICISC 2009. (2009)