Fast and Optimal Countermeasure Selection for Attack Defence Trees

Steve Muller[1,2,3(✉)], Carlo Harpes[1], and Cédric Muller[1]

[1] itrust consulting s.à r.l., Niederanven, Luxembourg
{steve.muller,harpes,cedric.muller}@itrust.lu
[2] University of Luxembourg, Luxembourg City, Luxembourg
[3] Telecom Bretagne, Cesson-Sévigné, France

Abstract. Risk treatment is an important part of risk management, and deals with the question which security controls shall be implemented in order to mitigate risk. Indeed, most notably when the mitigated risk is low, the costs engendered by the implementation of a security control may exceed its benefits. The question becomes particularly interesting if there are several countermeasures to choose from.

A promising candidate for modeling the effect of defensive mechanisms on a risk scenario are attack–defence trees. Such trees allow one to compute the risk of a scenario before and after the implementation of a security control, and thus to weigh its benefits against its costs.

A naive approach for finding an optimal set of security controls is to try out all possible combinations. However, such a procedure quickly reaches its limits already for a small number of defences.

This paper presents a novel branch-and-bound algorithm, which skips a large part of the combinations that cannot lead to an optimal solution. The performance is thereby increased by several orders of magnitude compared to the pure brute–force version.

Keywords: Attack-defence tree · Return On Security Investment · Optimal defences · Risk treatment optimisation · Branch and bound algorithm

1 Introduction

Several risk methodologies exist [1] that assist the risk assessor in identifying and handling risk, by providing exhaustive libraries of risk scenarios and/or defensive mechanisms. Those methodologies require organisations to conduct a risk assessment, which permits them to identify the risks that have to be mitigated. However, they do not prescribe in detail how organisations should put such a process into practise, leaving them enough freedom to choose an approach that fits their needs and requirements. There are several frameworks (such as ISO/IEC 27005 [2], IT-Grundschutz [3], MAGERIT [4] or EBIOS [5]) and commercial tools (such as TRICK Service[1]) that assist stakeholders in taking

[1] www.itrust.lu/products.

J. Großmann et al. (Eds.): RISK 2016, LNCS 10224, pp. 53–65, 2017.
DOI: 10.1007/978-3-319-57858-3_5

decisions for putting security controls in place. As for research, Attack–Defence Trees [6] constitute a visual and very intuitive technique for analysing a risk scenario in greater detail. They are a generalisation of ordinary attack trees [7]: The latter encode, in a tree structure, how an attack (the root node) can be achieved through intermediary attacks (its child nodes), so each branch adds further refinement the parent attack – see Fig. 1 for an example. In contrast, attack–*defence* trees also include the defensive mechanisms used to mitigate these attacks. More precisely, the associated defence nodes are appended as specially marked nodes to the attack nodes they protect from. These defences again face attacks on their own, which try to disable the countermeasures. In fact, attack–defence trees adopt the game-theoretic concept of two players, opponent and proponent, who alternately try to defeat each other [8]. Figure 2 depicts a simple attack–defence tree.

Recent research work by Gadyatskaya et al. [9] shows how attack–defence trees can be combined with existent libraries (such as ISO/IEC 27002 [10]) to determine the security controls an organisation shall implement. Indeed, when a given set of controls is implemented, it will reduce the overall risk, but also comes at a certain cost. Or, in other words, every selection of countermeasures comes with a certain return after a certain investment. However, if the investment outweighs the return, it is not sensible to mitigate the risk in the first place. The related optimisation problem consists in finding those controls that have the best return on investment.

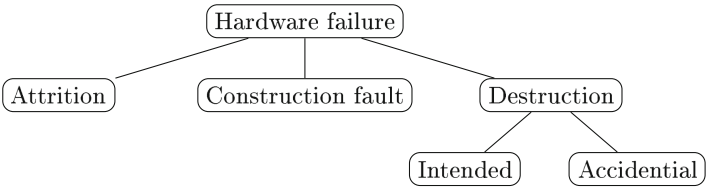

Fig. 1. A sample attack tree depicting the possible reasons of hardware failure.

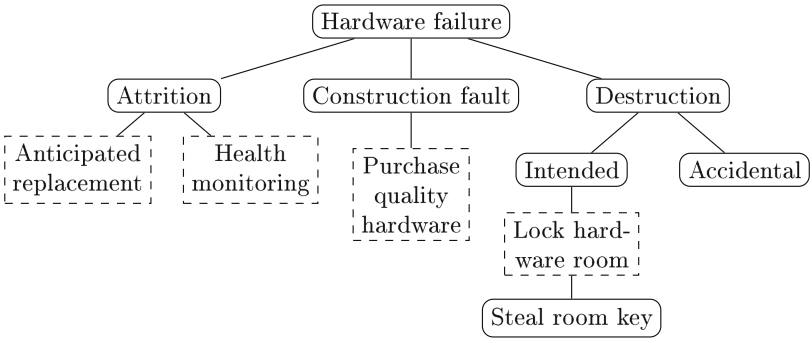

Fig. 2. A sample attack–defence tree, extending the example in Fig. 1. The dashed nodes are defence nodes.

The authors semi-automatically embed the security controls from ISO/IEC 27002 [10] as defence nodes into an existing attack tree. A simple brute-force program then iterates over all possible combinations of implementing those security controls, trying to find the strategy which maximises the return on investment. They have also developed a tool, ADTop, to demonstrate the work flow described in their paper. However, such an approach is very resource-intensive, and thus only works for very small input data. In this paper, we improve on their work and propose a faster and memory-saving algorithm for finding the set of security controls that minimises both risk and the security costs in an optimal way.

Other authors have proposed similar approaches and algorithms. Dewri et al. [11] propose a genetic algorithm that optimises a multi-objective function taking into account the attack probability and the implementation costs. However, they model defensive mechanisms as objects that mitigate an attack *completely*, which is far from reality. In contrast, Roy et al. [12] propose a model based on attack–defence trees and express the added-value of implementing a security control as return-on-investment, taking into account the risk reduction and the implementation costs. They propose a branch-and-bound algorithm, but it requires that at least one countermeasure is selected for each attack, which may not be necessarily sensible if the risk associated to that attack is already low.

This paper is organised as follows. Section 2 introduces the optimisation problem and the underlying model. Section 3 presents and deliberates the algorithm, as well as its performance. A real-world case study is used to substantiate the need for the algorithm in Sect. 4. A conclusion is drawn in Sect. 5.

2 The Optimisation Problem

2.1 Attack–Defence Trees

An attack–defence tree is defined [13] as a tree graph consisting of two kinds of nodes:

- **attack** nodes, characterised by a name and a success probability $p \in [0, 1]$;
- **defence** nodes, characterised by a name, an effectiveness $e \in [0, 1]$ and a cost $c \geq 0$.

The parameters have the following meaning:

- The **success probability** expresses the likelihood that the attacker succeeds in accomplishing the attack. If the node is a leaf, the success probability is part of the input. Otherwise, it is computed according to the rules defined below.
- The **effectiveness** expresses the degree (as a factor) to which the countermeasure reduces the attack probability. The value 0 indicates that it is entirely useless, 1 represents complete mitigation of the attack. The effectiveness is part of the input.
- The **cost** is expressed in financial terms and represents the cost engendered by the implementation of the defence. The cost is also part of the input.

The root node of an attack–defence tree is always an attack goal. Attack nodes can have subordinated attacks (that add more refinement) and defences (that defend against this attack). Defence nodes can only have subordinated attacks (that weaken the countermeasures).

For simplicity, this paper does not consider counter-attacks against defences. Thus, in the following, defences are assumed to have no subordinate child nodes.

Moreover, the set of child attacks can be 'disjunctive' or 'conjunctive', meaning that the parent attack consists of achieving *any* or *all* of the child attacks, respectively. Similarly, the set of child defences can be 'disjunctive' or 'conjunctive', meaning that *any* or *all* of the defences are required to protect from the attack, respectively.

All attacks and defences in the tree are assumed to be independent. This assumption is made to simplify the computations, and might not reflect reality. To take dependencies into consideration, more general models have to be considered, such as Bayesian networks – these are out of the scope of this paper, though.

2.2 Multi-purpose Defences

A defence can protect from several attacks, though possibly with a different effectiveness. For instance, digital e-mail signatures prevent content manipulation by third parties in a very effective fashion. At the same time, they verify the sender's identity and defend against impersonation attacks. However, the effectiveness is a bit lower in this case, since the recipient cannot be entirely sure that the sender is the real person he expected, for the latter could hack himself into that person's computer.

In this paper, defences are allowed to protect from multiple attacks, possibly with different effectiveness values. That is, if one decides to implement such a defence, and thus include it into the attack–defence tree, it will be appended to *all* applicable attacks.

2.3 Rules of Calculation

For an attack α, let $p(\alpha)$ denote its success probability. For a defence δ, let $c(\delta)$ denote its cost, and $e(\delta)$ its effectiveness.

When no defence mechanisms are present, and assuming that all attacks in the tree are independent, the following basic probability rules hold for a non-leaf attack node α.

$$p(\alpha) = \begin{cases} \prod_i p(i) & \text{if } \alpha \text{ is conjunctive} \\ 1 - \prod_i (1 - p(i)) & \text{if } \alpha \text{ is disjunctive,} \end{cases}$$

where i iterates over all child attack nodes of α. If a defence δ is in place, by definition of the effectiveness, it reduces the success probability by a factor

$$1 - e(\delta).$$

Similarly, if a set of defences Δ is in place, the success probability will be reduced by $1 - e(\Delta)$, where

$$e(\Delta) := \begin{cases} \prod_{\delta \in \Delta} e(\delta) & \text{if } \Delta \text{ is conjunctive} \\ 1 - \prod_{\delta \in \Delta} (1 - e(\delta)) & \text{if } \Delta \text{ is disjunctive,} \end{cases}$$

assuming that defences reduce the success probability independently from each other. So in summary, if a set Δ is implemented for an attack α, the recursive computation rule is given by

$$p(\alpha) = (1 - e(\Delta)) \cdot \begin{cases} \prod_{i} p(i) & \text{if } \alpha \text{ is conjunctive} \\ 1 - \prod_{i} (1 - p(i)) & \text{if } \alpha \text{ is disjunctive.} \end{cases} \tag{1}$$

The recursion ends at the leaf nodes, for which the probability is fixed and part of the input.

2.4 Optimisation Problem

Implementing a defence δ reduces the success probability, but also comes at a cost $c(\delta)$. It is not a-priori obvious whether it is profitable to implement a specific defence, because it could be wiser to select one or several others that come at a lower cost. The problem thus consists in finding those defences that reduce the success probability by a decent amount, but still come at a reasonably low cost.

In order to solve this multivariate optimisation problem, the Return On Security Investment (ROSI) is chosen as score function. It is defined as

$$\text{ROSI} := \underbrace{\text{impact} \cdot (\text{initial probability} - \text{final probability})}_{\text{return (risk reduction)}} - \underbrace{\text{sum of costs,}}_{\text{investment}}$$

where 'initial' and 'final' are understood to be before and after the implementation of all defences. A strategy is said to be optimal if it maximises the ROSI. Note that there are many ways to define the ROSI (see e.g. [14]); this definition was chosen because of its intuitive meaning and its simplicity.

Formally, denote the set of all available defences by D. An *assignment* is a function $x : D \to \{0, 1\}$ which states whether each defence δ shall be implemented ($x(\delta) = 1$) or not ($x(\delta) = 0$). The ROSI can mathematically be expressed as

$$\text{ROSI}(x) := \mathcal{I} \cdot (\mathcal{P}_0 - \mathcal{P}(x)) - \sum_{\delta \in D} x(\delta) \cdot c(\delta), \tag{2}$$

where $\mathcal{I}$ is the (constant) impact of the risk scenario, $\mathcal{P}(x)$ is the success probability of the attack–defence tree after implementing all defences with $x(\cdot) = 1$, and $\mathcal{P}_0$ is the (constant) success probability of the attack–defence tree (thus without any defences). The probabilities are calculated using the formula given in Sect. 2.3. The optimisation problem then reads as

$$\begin{aligned} \text{Find} \quad & x : D \to \{0, 1\} \\ \text{that maximises} \quad & \text{ROSI}(x). \end{aligned} \tag{3}$$

3 Branch and Bound Algorithm

The optimisation problem can be solved in several ways. One possibility would be to turn ROSI(x) as defined in Eq. (2) into a linear function and apply standard linear programming algorithms [15] on it. Such an approach has been proposed and described by Roy et al. [12]. While this technique works in theory, the size of the linear program exceeds the practical limits of feasibility very quickly. For the case study presented in Sect. 4 below, the linear program would have a size of 2^{16} variables.

The proposed algorithm is given in Algorithm 1 and basically enumerates all possible combinations of applying defences. However, it skips all sets of combinations that are known not to contain any solutions. Note that it will never skip a valid combination; this is proved below. The algorithm is invoked with $D_p := \emptyset$ and an empty map $x : \emptyset \to \{0, 1\}$. The attack–defence tree T, the set of defences D and the effectiveness values e remain constant throughout the algorithms.

Note that if it was not for lines 1–3, Algorithm 1 were just a recursive brute-force algorithm that tries out all possible ways of selecting defences. The innovation (and performance optimisation) lies in the lines 1–3.

The idea is to skip a recursion step whenever it is known that it cannot yield a viable combination of selecting defences. The skip criterion in line 1 originates from the following observation. Equation (1) reveals that whenever a defence is added to the attack–defence tree, the success probability of *any* attack node will either decrease or at least remain the same. In particular, the same is true for the global success probability of the tree.

Note that whenever the algorithm enters a recursion step, all non-processed defences are set to 'unselected'; this is assured by the start condition and line 16. Thus, all later (i.e. deeper) recursion steps will end up with a lower or equal overall success probability for the attack–defence tree. By consequence, once the probability is no longer sufficiently reduced to cover the costs (i.e., once a defence is no longer profitable), it will not be profitable for all later combinations, either. Which means that all subsequent combinations are known to be invalid *a-priori*, so they can be skipped.

3.1 Performance

The performance gain depends on the structure of the attack–defence tree. A stress test was conducted on a tree consisting of 81 nodes and 90 defences, each

Algorithm 1. Branch and bound algorithm BnBa

Input: Attack–defence tree T with attack nodes A
Input: Set of defences D
Input: Effectiveness values $e : A \times D \to [0, 1]$
Input: Set of already processed defences $D_p \subseteq D$
Input: Partial selection strategy $x : D_p \to \{0, 1\}$
Output: Selection strategy x_{opt} that maximises ROSI$(\cdot)$

1: **if** there is $\delta \in D_p$ that is no longer profitable (cf. Algorithm 2) **then**
2: **abort** current recursion step
3: **end if**
4: **if** $D_p = D$ **then**
5: $v \leftarrow$ **ROSI**(x)
6: **if** v is largest ROSI seen so far **then**
7: $x_{\text{opt}} \leftarrow x$
8: **end if**
9: **else**
10: $\delta \leftarrow$ any defence *not* in D_p
11: $D_p \leftarrow D_p \cup \{\delta\}$
12: ' Try selecting the defence
13: $x(\delta) \leftarrow 1$
14: BnBa(T, D, e, D_p, x)
15: ' Try not selecting the defence
16: $x(\delta) \leftarrow 0$
17: BnBa(T, D, e, D_p, x)
18: ' Remove δ again; this allows the re-use of D_p among all recursive calls
19: $D_p \leftarrow D_p \setminus \{\delta\}$
20: **end if**

of which is applied to every attack. The resulting attack–defence tree has thus $81 \cdot 90 = 7290$ defence nodes. Note that in a concrete case, not every defence would be applicable for every attack, and by consequence, the problem would be simpler. The effectiveness values $e : A \times D \to [0, 1]$ were chosen randomly.

If one comments out lines 1–3 in Algorithm 1, one obtains a pure brute-force algorithm that tries out all 2^{90} combinations. Executing it for the first 2^{20} combinations took 107.42 s in our implementation; so it would need $1.27 \cdot 10^{23}$ s ($4 \cdot 10^{15}$ years) to finish. On contrast, the optimised variant terminated within 895 s (15 min), having evaluated only $1,748,272$ combinations (which is approximately a 10^{-21} part).

Algorithm 1 can be implemented in such a way that it uses constant memory in the course of its execution. This can be achieved by using a `stack` data structure for D_p and a fixed-size array for x; both D_p and x are shared among all recursive calls of the algorithm. In our implementation the memory usage was approximately 20 MiB for the tree described above.

Algorithm 2. Determine if a defence is profitable

Input: Defence δ
Input: Cost $c(\delta)$ of defence δ
Input: Impact $\mathcal{I}$ of risk scenario
Input: Partial selection strategy $x : D_p \to \{0, 1\}$
Output: **true** if δ is profitable, **false** otherwise

if $x(\delta) = 0$ **then**
    return true
else
    ' Extend x to all of D
    $x(\delta') \leftarrow 0$ for all $\delta' \in D \setminus D_p$
    $x(\delta) \leftarrow 0$
    $v_0 \leftarrow \text{ROSI}(x)$
    $x(\delta) \leftarrow 1$
    $v_1 \leftarrow \text{ROSI}(x)$
    ' δ is profitable iff the residual risk is lower when δ is implemented
    if $v_1 \cdot \mathcal{I} + c(\delta) < v_0 \cdot \mathcal{I}$ **then**
        return true
    else
        return false
    end if
end if

The tests were conducted on a standard laptop with a i7-6700HQ processor (2.6 GHz). Our implementation of the algorithm ran on a single core, although it can be modified in such a way that it supports multi-threading, as well.

4 Case-Study

The methodology presented in [9], together with the new Algorithm 1, is used to determine those ISO 27002 [10] security controls that have the largest added-value for the 'ÉpStan' project.

ÉpStan, which is short for *Épreuves Standardisées* (standardised exams), is Luxembourg's national programme to monitor the quality of the educational system of secondary school. To achieve this, standardised exams are conducted in selected classes all over the country, and the results are analysed to spot topics that are not well covered by the school programme.

Since the tests are meant to rate the educational system, rather than the individual students' performance, the results should under no circumstances be linked to the individuals. There are four parties involved in the process:

- The *Government* provides the standardised exams.
- The *schools* organise and conduct the exams.
- The *University of Luxembourg* is responsible for evaluating the results.

– *itrust consulting* acts as a trusted third party and pseudonymisation service between the Government and the University of Luxembourg. Its role is to assure that neither of them can link results to an individual student. This is achieved by issuing a pseudonym for each student, which is used by the schools to exchange the exam results with the University. itrust consulting never obtains any exam result.

The process is designed in such a way that neither the University, nor the Government, nor itrust consulting can link exam results to individual students. Although every entity only knows part of the necessary information, an attacker could get (legally or not) data from multiple entities, and reconstruct the link between result and student.

A brainstorming session led to the identification of an exhaustive list of attack scenarios, all of which have been encoded in an attack tree consisting of 81 attack nodes. Figure 3 shows a small excerpt.

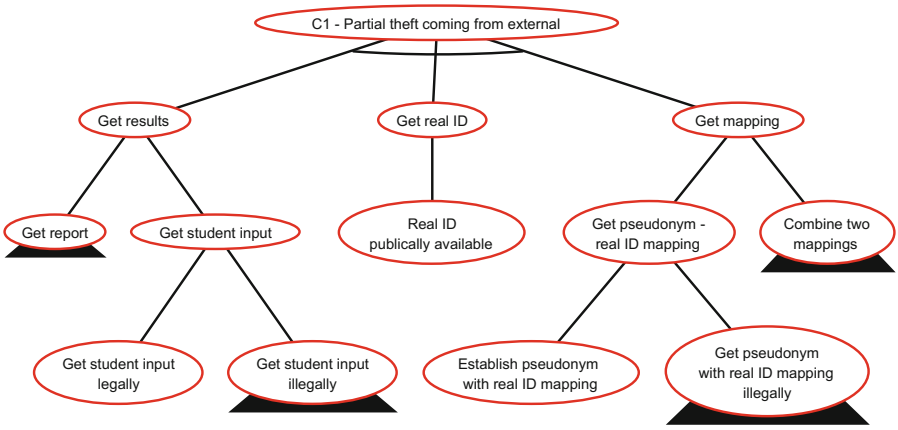

Fig. 3. An excerpt of the attack tree for the risk scenario where an attack can link exam results to a student.

The second step consisted in determining the ISO 27002 [10] controls that reduce the success probability of some of the identified attacks. In total 16 controls were retained. Moreover, the effectiveness was estimated in a brainstorming process for each of the retained defences and each of the applicable attacks. The resulting effectiveness matrix $e : A \times D \to [0, 1]$ had 58 non-zero values and is depicted in Table 1.

ADTop (see [9]) required 54.2 s and over 1 GiB of memory to find the optimal attack–defence tree. A C# implementation of Algorithm 1 proposed the same set of defences within 0.36 s, having tried out 12,496 combinations (19% out of the 2^{16} possible). The memory usage was 20 MiB. An excerpt of the full attack–defence tree is depicted in Fig. 4.

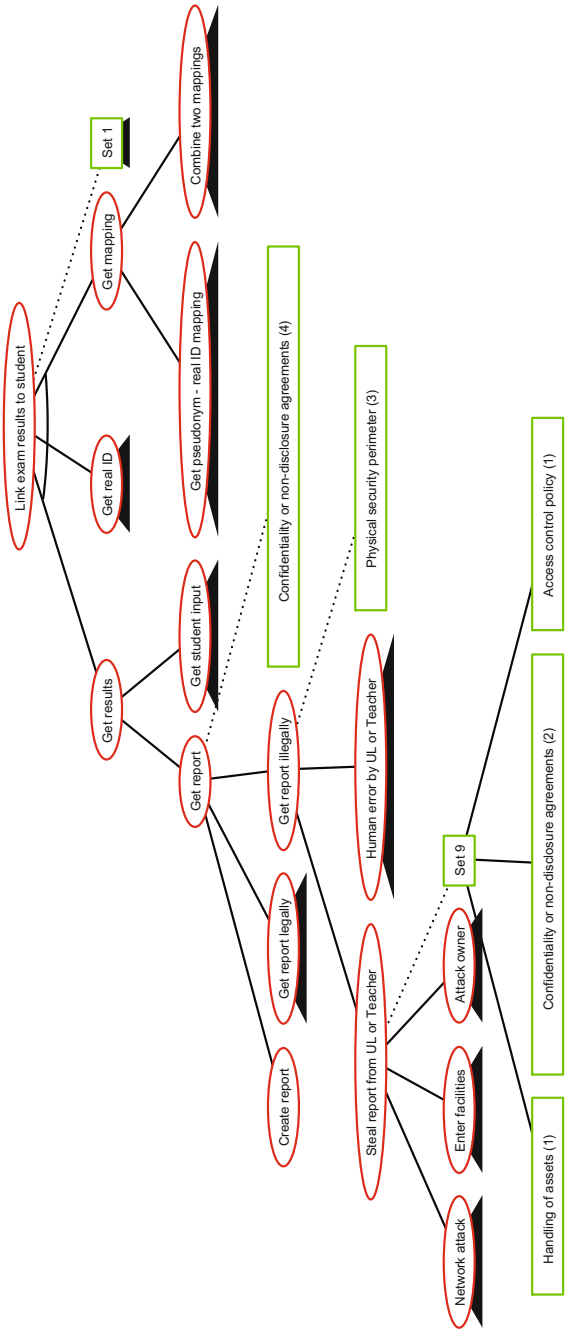

Fig. 4. An excerpt of the attack–defence tree with security controls appended.

Table 1. An excerpt of the effectiveness matrix, which has (16 defences × 81 attacks). Values indicated are in %.

	Confidentiality or non-disclosure agreements	Access control policy	Network controls	Physical security perimeter	Segregation of duties	Handling of assets	Password management system	Restrictions on software installation	Removal or adjustment of access rights	Secure disposal or re-use of equipment	Management of technical vulnerabilities	Event logging	Administrator and operator logs	Change management	Capacity management	Information security roles and responsibilities
Combine two mappings		50			5											
Human error by UL or Teacher	20															
Human error by TTP	20															
Steal student input from EpStan party	5	10			5	2										
Get pseudonym - real ID mapping		1			10	2			1							
Get student input illegally	5			5												
Network attack			50									2				
Steal from TTP or School or Teacher	5	10			5	2							2			
...																
Steal pseudonym from UL or TTP	5	10			5	2			0.5							
Human error by TTP or School or Teacher	20															
Steal report from UL or Teacher	5	10			5	2										
Get or steal temp ID from Post						2	20									
Link exam results to students			5	10		1		8	1		5	1	1	2	2	0.01
Get real ID	2	10	5			2										
Get report legally	5				5	2										
Steal item	2			20												

Out of the original 16 defences, the following 9 have been proposed for implementation by the algorithm.

- Handling of assets (Sect. 8.2.3)
- Password management system (Sect. 9.4.3)
- Physical security perimeter (Sect. 11.1.1)
- Capacity management (Sect. 12.1.3)
- Management of technical vulnerabilities (Sect. 12.6.1)
- Restrictions on software installation (Sect. 12.6.2)
- Network controls (Sect. 13.1.1)
- Confidentiality or non-disclosure agreements (Sect. 13.2.4)
- Access control policy (Sect. 9.1.1)

This solution reduces the success probability of the global attack from 29% to 6.5% and has a ROSI of 10031 EUR.

5 Conclusion

Gadyatskaya et al. [9] have shown how attack–defence trees can be used to model risk reduction engendered by a library of security controls. Since the implementation of defensive mechanisms comes at a cost, it is not *a prori* clear which controls to prefer over which ones. The authors determine the optimal defence strategy by literally processing all combinations of selecting security controls and computing the Return On Security Investment (ROSI) for each of them.

The tool presented in [9], 'ADTop', reaches its feasible limits at 16 defences. However, *any* pure brute-force program would have a practical limit of 40 defences. Indeed, if an evaluation of a single combination takes 1 ms, then iterating over all 2^{40} combinations will already take approximately 13 days (growing exponentially with the number of defences).

This paper improves on the latter work by presenting a memory-efficient algorithm which skips some of the unnecessary computations. This method experimentally decreases the running time of the algorithm on large trees (81 attacks, 90 defences) from several hundred years to several hours. While the technique works specifically for the ROSI function, it can be generalised to other, similar score functions, as well.

The improved algorithm has been applied in a case study in order to highlight the performance boosts. The case study deals with determining the optimal set of ISO 27002 countermeasures that shall be implemented for a pseudonymisation service, and uses an attack–defence tree consisting of 81 attacks and 16 unique defences. Compared to 'ADTop', the new algorithm reduces the memory usage from over 1 GiB to 20 MiB, and the execution time from nearly a minute to less than a second.

Acknowledgements. This work was supported by the Fonds National de la Recherche, Luxembourg (project reference 10239425) and the European Commission's Seventh Framework Programme (FP7/2007-2013) under grant agreement number 318003 (TREsPASS).

References

1. Giannopoulos, G., Filippini, R., Schimmer, M.: Risk Assessment Methodologies for Critical Infrastructure Protection, Part i: A State of the Art. Publications Office of the European Union, Luxembourg (2012)
2. International Organization for Standardization, ISO/IEC 27005 - information technology - security techniques - information security risk management (2011)
3. Bundesamt für Sicherheit in der Informationstechnik (BSI), IT-Grundschutz

4. Amutio, M.A., Candau, J., Mañas, J.: Magerit-version 3, methodology for information systems risk analysis and management, book I - the method, Ministerio de administraciones públicas (2014)
5. Secrétariat général de la défense nationale, Ebios-expression des besoins et identification des objectifs de sécurité (2004)
6. Kordy, B., Mauw, S., Radomirović, S., Schweitzer, P.: Attack–defense trees. J. Logic Comput. **24**(1), 55 (2014). doi:10.1093/logcom/exs029
7. Schneier, B.: Attack trees. Dr. Dobb's J. **24**(12), 21–29 (1999)
8. Kordy, B., Mauw, S., Melissen, M., Schweitzer, P.: Attack–defense trees and two-player binary zero-sum extensive form games are equivalent. In: Alpcan, T., Buttyán, L., Baras, J.S. (eds.) GameSec 2010. LNCS, vol. 6442, pp. 245–256. Springer, Heidelberg (2010). doi:10.1007/978-3-642-17197-0_17
9. Gadyatskaya, O., Harpes, C., Mauw, S., Muller, C., Muller, S.: Bridging two worlds: reconciling practical risk assessment methodologies with theory of attack trees. In: Kordy, B., Ekstedt, M., Kim, D.S. (eds.) GraMSec 2016. LNCS, vol. 9987, pp. 80–93. Springer, Cham (2016). doi:10.1007/978-3-319-46263-9_5
10. International Organization for Standardization, ISO/IEC 27002 - information technology - security techniques - code of practice for information security management (2013)
11. Dewri, R., Poolsappasit, N., Ray, I., Whitley, D.: Optimal security hardening using multi-objective optimization on attack tree models of networks. In: Proceedings of the 14th ACM Conference on Computer and Communications Security, pp. 204–213. ACM (2007)
12. Roy, A., Kim, D.S., Trivedi, K.S.: Scalable optimal countermeasure selection using implicit enumeration on attack countermeasure trees. In: IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012), pp. 1–12. IEEE (2012)
13. Kordy, B., Mauw, S., Radomirović, S., Schweitzer, P.: Foundations of attack–defense trees. In: Degano, P., Etalle, S., Guttman, J. (eds.) FAST 2010. LNCS, vol. 6561, pp. 80–95. Springer, Heidelberg (2011). doi:10.1007/978-3-642-19751-2_6
14. Gordon, L.A., Loeb, M.P.: The economics of information security investment. ACM Trans. Inform. Syst. Secur. (TISSEC) **5**(4), 438–457 (2002)
15. Luenberger, D.G.: Introduction to Linear and Nonlinear Programming, vol. 28. Addison-Wesley Reading, MA (1973)