Proceedings of the Federated Conference on Computer Science and Information Systems pp. 527–536 DOI: 10.15439/2020F21 ISSN 2300-5963 ACSIS, Vol. 21

Developing Defense Strategies from Attack Probability Trees in Software Risk Assessment

Marko Esche
Physikalisch-Technische Bundesanstalt,
Abbestraße 2-12, 10587 Berlin, Germany
Email: marko.esche@ptb.de

Federico Grasso Toro
Federal Institute of Metrology METAS,
Lindenweg 50, 3003 Bern-Wabern, Switzerland
Email: federico.grasso@metas.ch

***Abstract*—Since the introduction of the Measuring Instruments Directive 2014/32/EU, prototypes of measuring instruments subject to legal control in the European Union must be accompanied by a risk assessment, when being submitted for conformity assessment. Taximeters, water meters, electricity meters or fuel pumps form the basis for the economic sector usually known as Legal Metrology, where the development towards cheaper all-purpose hardware combined with more sophisticated software is imminent. Therefore, a risk assessment will always have to include software-related issues. Hitherto, publications about software risk assessment methods lack an efficient means to derive and assess suitable countermeasures for risk mitigation. To this end, attack trees are used in related research fields. In this paper, defense probability trees are derived from attack probability trees, well-suited to the requirements of software risk assessment and used to identify optimal sets of countermeasures. The infamous Meltdown vulnerability is used to highlight the experimental application of the method.**

I. Introduction

LEGAL Metrology covers measurements and measuring instruments used as a basis for economic transactions. In this paper, a method to assess the risks associated with these instruments is used to derive suitable countermeasures to mitigate identified risks. The Measuring Instruments Directive (MID) [1], whose aim is to establish trust in measurements for users and customers of such instruments, lays down the basic procedures for putting measuring instruments on the market. Trust in measurements plays a significant role, since the Legal Metrology sector generates an annual turnover of around 500 billion Euros for the European Union market [2].

As a first step, conformity to so-called essential requirements, laid down by Annex 2 of the MID, shall be demonstrated by the manufacturer with the help of a conformity assessment procedure. These essential requirements encompass physical properties of the measuring instrument, such as climatic operating conditions and electromagnetic compatibility testing, as well as information technology requirements on software and data protection.[1]

Over the past decade, it has become apparent that measuring instruments are going through a transformation process from simple stand-alone devices with integrated hardware and software to complex distributed systems, which use cheaper, less sophisticated hardware with more complex software. Therefore, the risk assessment to be supplied by manufacturers for conformity assessment of a prototype [1] has to specifically address the risks related to software, as well as its inherent hardware risks.

As an aid for manufacturers, PTB has published a method for software risk assessment, specifically tailored for the use in Legal Metrology [2] based on ISO/IEC 27005 [3] and ISO/IEC 18045 [4]. With the help of generic assets to be protected, derived from the MID, the method allows objective comparisons of different instruments produced by different manufacturers. Following definitions from ISO/IEC 27005, the method calculates risk as the combination of the impact produced by the realization of certain threats to assets and of the probability of occurrence of each threat. The technical steps to realize a threat are normally summarized in so-called attack vectors. An attempt at providing standardization to derive attack vectors is described in [5]. The approach is based on the attack tree concept used in related fields of research, such as the design of cryptographic protocols and access control [6].

In the present paper the attack probability trees (AtPTs) presented in [5] are explained in detail and extended towards defense probability trees (DePTs), as a formal method to derive optimal countermeasures to be used for risk mitigation. After this brief introduction, the remaining paper is structured as follows: In Section II, a literature overview is presented, which sketches a brief history of attack trees, explaining their basic functionality and applications. Section III provides a summary of the original method and recapitulates the concept and the applicability of AtPTs. Then, Section IV addresses how to find optimal sets of countermeasures, by means of DePTs derived from AtPTs. An experimental application of the method, focused on the Meltdown vulnerability [7] and its implications for Legal Metrology, is presented in Section V. Finally, Section VI provides a summary of the paper and potential further work.

II. Literature Overview

According to the international standard ISO/IEC 27005 [3] the risk assessment process can formally be divided into three

[1]One conformity assessment body within the European Union is the *Physikalisch-Technische Bundesanstalt* (PTB), Germany's national metrology institute. One conformity assessment body that follows the MID as a consequence of the bilateral agreements between Switzerland and the European Union, is the Federal Institute of Metrology METAS.

phases: 1) The risk identification phase: It normally starts with the definition of certain assets to be protected. A derivation of assets applicable to Legal Metrology was presented in [2] and a short example is supplied in Section III. 2) The risk analysis phase: Based on the identified assets, threats are formulated, constituting an invalidation of a specific security property of an asset by an attacker with an associated impact. The technical steps needed to realize a threat, commonly referred to as attack vectors, are also identified during this phase. 3) The evaluation phase: The final stage of the risk assessment is the evaluation of the risk associated with a threat. The evaluator decides whether the estimated risk is tolerable or if countermeasures need to be implemented.

The following example from IT security clarifies these three phases: If a cryptographic key on a computer must be protected against retrieval by an attacker, during the first phase (risk identification), the key becomes one asset to be protected with confidentiality, as the associated security property. One way to protect such a key would be the read/write permissions of the operating system, which - under normal conditions - can only be changed by using the administrator's credentials. During the second phase (risk analysis), the threat can be formulated: "By guessing the correct administrator password - an action which corresponds to one possible attack vector - an attacker would be able to modify the read/write permissions for the cryptographic key and retrieve it without being detected." Among other things, the likelihood of said attack vector would then depend on the window of opportunity to access the computer and on the strength of the chosen password. In the final phase, sc. risk evaluation, the calculated risks are prioritized and a cut off point for the risk assessment is defined. For all unacceptably high risks, countermeasures are then selected and implemented to repeat the assessment process until all risks are classified as tolerable. The theoretical framework for the selection of countermeasures is described in Section III and illustrative examples can be found in Section V.

This paper focuses on the efficient graphical and logical representation of these vectors and on the identification and selection of its countermeasures. Originally, attack trees were used by human evaluators to illustrate and identify vulnerabilities of a known system by graphical means. However, these trees also have a number of mathematical properties that make them well-suited for automatic processing and evaluation. The following sub-sections provide the foundations of attack trees and their applications on risk assessment of software.

A. Foundations of Attack Trees

In [6], a detailed general introduction to attack trees and their potential applications is given. According to Mauw and Oostdijk, the most basic properties of any attack tree can be summarized as follows: While the root node of such a tree constitutes an attacker's main goal, its child nodes can be seen as refinements thereof, which need to be achieved in order to reach said goal. Following this interpretation, the leaves of an attack tree constitute atomic attacks, for which no further refinement is possible. An exemplary tree that only consists of seven nodes is given in Figure 1.

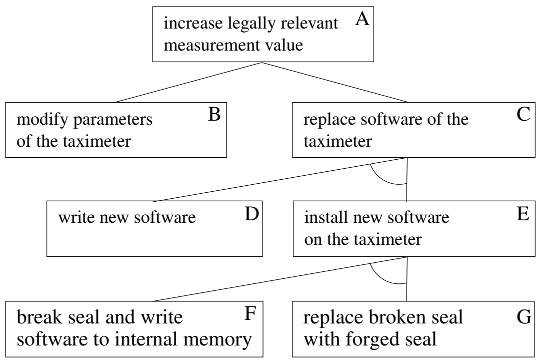
increase legally relevant measurement value A
modify parameters of the taximeter B
replace software of the taximeter C
write new software D
install new software on the taximeter E
break seal and write software to internal memory F
replace broken seal with forged seal G

Fig. 1. Simple illustration of an attack tree that shows how the calculated fare/measurement value of a taximeter may be manipulated.

In the example, an attacker's possible strategies to manipulate the fare calculated by a taximeter are illustrated. Before exploring the meaning of the shown tree, it is necessary to explain the specifics of its graphical representation: Child nodes are always logically connected to either form an AND- or an OR-expression. The AND-statement is illustrated by an arc connecting the respective child nodes and indicates that all of these need to be implemented to achieve the attack associated with the parent node. On the other hand, if child nodes represent alternative ways to reach the parent objective, they are connected via an OR-statement, in which case no arc is drawn. Mauw and Oostdijk [6] interpret the two possible connections between nodes as 'conjunctive aggregation' and 'disjunctive refinement (choice)'. Certainly, there is no guarantee that an attack tree will be a binary tree. However, if more than two child nodes are identified, they can always be transformed into a binary structure by combining pairs of them into subgoals until only two child nodes remain. The examplary attack tree given in Figure 1 states that the fare can either be manipulated (node A) by changing the parameters of the taximeter (node B) or by replacing its software (node C). Other possible alternatives are not discussed here, since this example simply aims to illustrate the operation modes of attack trees. Therefore, at the root node a simple OR-connection can be found. Consequently, replacing the software requires at least two steps: 1) writing a new software (node D) and 2) installing it on the instrument (node E). Both steps need to be implemented for the attack to be executed successfully, which is expressed by an AND-statement, see Figure 1. It should be noted that AND-connected attacks do not necessarily need to be implemented simultaneously, as is the case in the illustrated scenario. The process of installing the software can once more be sub-devided into two AND-connected tasks: 1) opening the taximeter (node F) to write new software to its memory and 2) forging a new seal afterwards (node G). It can be seen that the combination of two nodes into a summary node has no influence on the mathematical properties of the local sub-tree, such as likelihood of occurrence [6]. Therefore, it is the evaluator's choice to limit the number of refinements of an attack as she sees fit. In practice, a node needs no further refinement if the associated attack constitutes a simple technical task with a known scope and easily determinable properties.

In addition to providing basic means of describing and manipulating attack trees, Mauw and Oostdijk introduce the concept of predefined characteristics of a node, e.g. possibility, cost, equipment needed. They show that the attributes of any parent node can be determined by combining the information associated with the respective child nodes. They highlight the point that these rules will be specific to the application, although they can usually be determined directly from the attributes and their properties. These rules work directly from the leaves towards the root without having to resort to trace-backs. After the application of these rules, the result is a set of attribute values, either of the root node itself or of a chosen sub-tree. In the latter case, the sub-tree may either reflect a set of promising attacks or contain information, such as its internal structure, not directly represented by the values of its attributes. It is important to note that there is no requirement for any node to only exist once within a tree [6]. Instead, nodes may have multiple copies whose attributes are linked; therefore, a change in one part of an attack tree can also affect otherwise unconnected branches. Mauw and Oostdijk also introduce the attack suite as a set of attacks that can be used to realize a goal without having to address the logical structure of the summarized sub-trees and their nodes. They show that attack trees with different structural representations may, nevertheless, contain the same logical information.

Concerning attack tree transformations, Mauw and Oostdijk formulate two rules: 1) 'associativity of conjunction' and 2) distributivity of 'conjunction over disjunction'. 'Associativity of conjunction' basically states that any sub-tree can be moved to its parent node if the root of the sub-tree is the only child node of the parent. In this case, parent node and child are logically identical. The second rule, which addresses the distributivity of 'conjunction over disjunction', states that any node with two separate sub-trees corresponds to two copies of the same node with one sub-tree each. Mauw and Oostdijk [6] provide proofs for both transformation rules.

The following additional remarks by Mauw and Oostdijk on the attributes of attack trees will be used later in Section III: The value of an attribute can only be determined if the semantics of the tree are known. With this knowledge the value of the attribute can be estimated from the properties of the equivalent attack suite. Additionally, the authors state that the attributes of a node can only depend on the attributes of its child nodes alone.

B. Threat Risk Analysis for Cloud Security based on Attack-Defense Trees

The attack trees discussed so far only focus on strategies an attacker might follow. In [8], Wang, Lin, Kuo, Lin and Wang introduce a new derivative of such trees that also offers the possibility to model defensive strategies. These Attack-Defense Trees (ADT) are specifically tailored to describe the attack profile and calculate the associated attack cost. Both can then be used to choose appropriate countermeasures even for complex attacks. This process is usually referred to as Threat Risk Analysis (TRA) which takes vulnerability information and attack profiles as input. Within the scope of a TRA, there are both the estimation of the impact of a successful attack and a precise description of attack progression. The combination of both allows the user of the method to develop adequate contermeasures/defense strategies. It should be observed that, when trying to describe all possible attack paths at a desired level of detail, attack trees quickly become complex and difficult to handle. Wang, Lin, Kuo, Lin and Wang therefore claim that expressing both attack and defensive strategies in the same tree is usually too complex and beyond the scope of the tree. In contrast to attack trees, which are used to model systemic weaknesses, protection trees migrate weaknesses and thus have the potential to help to identify protective strategies. Section III gives more theoretical background on this, while Section V provides an example of weakness migration and defense strategy selection. As explained in [8], it was usually assumed that an attacker has complete information about the system to attack and would always select the easiest available attack path. In reality, the attacker may, however, act upon an incomplete set of facts, which will affect her ability to select the easiest route. This needs to be taken into account when choosing appropriate countermeasures. Wang, Lin, Kuo, Lin and Wang then supply equations for estimating probability of occurence and other metrics for both AND- and OR-connections between attack nodes. Since the ADT is supposed to be used for both attack and defense modeling these metrics do not only include probability of success, attack cost and impact, but also revised attack cost and revised impact for the countermeasure stage, adding a forth and final phase (risk mitigation) to the previously described risk assessment process. Similar metrics are used in Section III. Once again, all metrics are estimated for the leaf nodes first and then propagated up the tree towards the root node. Whenever assessing risks of a new system, the first step consists of identifying possible vulnerabilities. To this end, public databases are one important source of information. The second step of an assessment is the collection of information on recognized attacks. This includes identification of attack vectors or, more generally, of means to realize a threat by exploiting a known vulnerability. In a third step, an ADT is built, including all identified vulnerabilities which could facilitate an attack. To construct the tree and fill it with information, rules for the transfer of the metrics mentioned above are postulated. The fourth step then consists of the systematic evaluation of the ADT. Since data on past incidents may not always be available, the calculated probability of occurrence is always affected by a level of uncertainty. Nevertheless, it is stated that there is a deterministic transfer function between attack cost and defense cost, influenced by security policies, procedures, equipment and training of personnel. The final step for the ADTs is to establish adequate countermeasures for each attack. Despite the general applicability of their concept, Wang, Lin, Kuo, Lin and Wang state that the final step would need to be individually tailored for each new scenario, since the countermeasures solely depend on the available vulnerabilities.

C. Automated Generation of Attack Trees

Mauw and Oostdijk [6] showed that several different graphical representations may exist for one logical attack tree. Since the layout of the tree itself is thus subject to the evaluator's decisions, an automated approach could ensure that all attack trees adhere to the same design principles. One such approach was presented by Vigo, Nielson and Nielson [9]. They overcome the stated problem by inferring attack trees from the process logic otherwise used to describe them. According to Vigo, Nielson and Nielson both the scientific community and the wider public in general profit from attack trees since they are easily quantifiable as well as easily comprehensible. In the described implementation, the root node again reflects a threat to be implemented and child nodes describe sub-goals to be combined to realize the threat. To build the attack tree the attack process is first translated into propositional formulae from which the tree is then automatically inferred. The result subsequently does not suffer from human interpretation errors and is thus easily reproducible. Once the attack tree has been derived, the leaves, which are interpreted as atomic attacks, are labeled with individual costs that propagate up the tree. In line with the process-oriented approach, attack and defense actions are seen as communication processes from signal theory and are treated accordingly. From these, the cheapest set of atomic attacks is selected, corresponding to the most likely attack path, and the resulting attack cost is calculated. This idea is reused in Section IV to derive optimal defense strategies.

III. Software Risk Assessment in Legal Metrology

The software risk assessment method, used here to construct a method for countermeasure identification and selection, was originally published in [2]. The method is currently used by both examiners and manufacturers during conformity assessment of measuring instruments within the EU. A number of modifications and additions, to this method have been published [5]. In this section these publications are quickly summarized to lay the groundwork for the countermeasure derivation introduced in Section IV.

A. Fundamentals of Risk Assessment Method

In line with the formal requirements and definitions for risk assessment laid down in the international standard ISO/IEC 27005 [3], risk can be defined as a 'combination of impact and probability of occurrence attributed to a threat'. A threat is seen as 'an adverse action carried out by an attacker/threat agent upon the security properties of an asset'. In principle, the standard offers examiners a choice between quantitative and qualitative interpretations of the associated terms. Here, the quantitative approach is used to make results more easily reproducible and algorithmically processable.

TABLE I
Mapping between TOE-resistance and attack probability score [2].

Sum of Points	TOE Resistance	Probability Score
0-9	No rating	5
10-13	Basic	4
14-19	Enhanced Basic	3
20-24	Moderate	2
> 24	High	1

As a first building block, assets derived from legal requirements in Annex I of the MID are detailed [2]. Within this paper, measurement data with their associated security properties integrity and authenticity will be used as an example. Nevertheless, the following observations are applicable to all other assets as well. Availability of data is not required, since in absence of measurement data no commercial transaction can take place and no financial damage can be inflicted. Subsequently, an example for a possible threat could be formulated as follows:

An attacker falsifies the authenticity of measurement data.

In line with the definition from ISO/IEC 27005, values for impact and probability of occurrence are needed to calculate a numerical risk score. Here, only two impact values are used ($\frac{1}{3}$ if a single measurement is affected and 1 if all future or past measurements are affected). To evaluate the probability of occurrence, technical details (so-called attack vectors) needed to implement a threat are evaluated according to the vulnerability analysis described in ISO/IEC 18045. For example, guessing an administrator's password would be one attack vector to obtain access to protected operating system features. The evaluation of all possible attack vectors for an attack by means of the ISO/IEC 18045 vulnerability analysis consists of assigning point scores in five different categories: 1) time required; 2) expertise required; 3) knowledge needed about the target of evaluation (TOE); 4) window of opportunity; and 5) equipment needed to implement the attack. An expertise score of 0, for instance, reflects the fact that the attack can be carried out by a layman. If an expert is needed instead, the score is set to 6. Incidentally, the very same scores can also be used to describe the cost of implementing a countermeasure to an attack since attacker and defender have to perform similar tasks, see Section V. More detailed examples that address all five scores may be found in Section V of this paper and in ISO/IEC 18045 [4]. Once the scores for all five categories have been assigned, the sum score is calculated. In the original ISO/IEC 18045 this sum score with a theoretical upper limit of 51 points is mapped to a TOE resistance between 'no rating' and 'high'. It follows that an attack is less likely to happen if the sum score is higher. As described in [2], this resistance can be transformed into an attack probability score between 1 and 5, where 5 corresponds to a high likelihood, see Table I. The probability score is then multiplied with the determined impact to calculate the numerical risk score, which is again in the range between 1 and 5. An overview of the entire risk assessment workflow is given in [2].

B. Extension of Risk Assessment method for Attack Probability Trees

In [5] the attack tree concept from Mauw and Oostdijk was extended by augmenting the attack nodes with attributes such as time and expertise detailed above. The resulting attack probability trees (AtPTs) both represent the attack logic and the probability of occurrence (and subsequently risk) associated with a threat. This means that each attack vector is no longer evaluated individually, but only the atomic attacks at the leaf nodes are assessed. This reduces the possibility for misjudging an attack and makes it possible to re-use atomic attacks for different threats.

To propagate the attributes up the tree a number of rules specifically tailored for the characteristics of each attribute were introduced. Since these rules are used extensively in the experimental example in Section V, the following description is needed:

- Time
  - **AND**: Time representation in point scores is logarithmic (1 for more than a day, 2 for one to two weeks, 19 for half a year). Adding up times for two attacks can, therefore, be approximated by selecting the maximum of the two.
  - **OR**: The time score connected to the smaller sum-score is chosen.
- Expertise
  - **AND**: Normally, the maximum of both scores is chosen. Should expertise in both hardware and software (HW and SW) be needed, scores are added with a maximum value of 8, see ISO/IEC 18045.
  - **OR**: The expertise score connected to the smaller sum-score is chosen.
- Knowledge of the TOE
  - **AND**: The maximum of both knowledge scores is chosen.
  - **OR**: The knowledge score connected to the smaller sum-score is chosen.
- Window of opportunity
  - **AND**: A smaller window of opportunity (higher score) for one node will also affect the other node. Therefore, the maximum is selected.
  - **OR**: The window of opportunity score connected to the smaller sum-score is chosen.
- Equipment
  - **AND**: The maximum of both equipment scores is chosen unless equipment from different areas is required (HW or SW), in which case the scores are added with a maximum of 9 according to ISO/IEC 18045.
  - **OR**: The equipment score connected to the smaller sum-score is chosen.

IV. SELECTING COUNTERMEASURES

In the context of this paper, a countermeasure shall be any technical modification of a measuring instrument or organizational measure that results in the reduction of the overall risk associated with the instrument below a predefined threshold. However, finding a set of countermeasures to prevent an attack from happening is more complex then simply countering each individual atomic attack.

A. Connection between AtPTs and Logic Networks

In the case of an AND-statement, increasing the attack cost, i.e. the sum score of one leaf node, always affects the sum score of the parent node. For OR-connected nodes, modification of one node may be sufficient, if the other node's sum score is already above a predetermined level and the associated risk is subsequently low enough. In other cases, however, it may be necessary to counter both child nodes of an OR-connected node to increase the cost associated with the parent node sufficiently, see Figure 2.

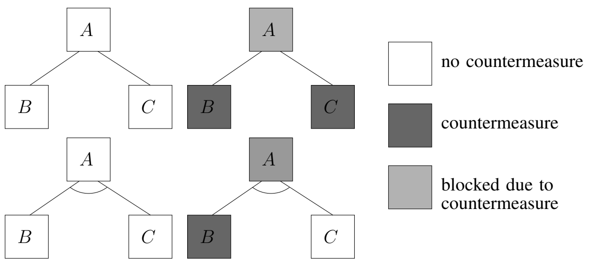

Fig. 2. Two seperate kinds of atomic AtPTs exist which correspond to OR-statements (*above*) and AND-statements (*below*) in boolean logic. For an OR-statement it may be necessary to counter both leaf attacks i.e. to increase their individual costs to mitigate the threat while one blocked leaf always has the same affect for an AND-statement.

It should be noted that the two atomic trees correspond directly to the fundamental building blocks of any logic network. It follows that even more complex trees can be converted into logic networks. This is the inverse principle of the method described in [9]. Of course, the interpretation of AtPTs as logic networks, see Figure 3, lacks all the information concerning individual attributes and the probability of occurrence or risk. However, this simple Boolean view of an AtPT constitutes a practical step towards identifying potential sets of countermeasures to be evaluated. To derive practical countermeasures to be implemented, any AtPT can first be transformed into the equivalent Boolean equation. The corresponding term for the exemplary tree given in Figure 1 is given in Equation (1).

$$A = B \vee C = B \vee (D \wedge E) = B \vee (D \wedge (F \wedge G)) \tag{1}$$

It should be noted that nodes C and E disappear in this representation since they only act as intermediate summary nodes. The final equation only contains the root and leaf nodes. By calculating the logical inverse of Equation (1) the Boolean expression of the Defense Probability Trees (DePT) can be found, see Equation (2).

$$\begin{aligned}\overline{A} &= \overline{B \vee (D \wedge (F \wedge G))} = \overline{B} \wedge \overline{(D \wedge (F \wedge G))} \\ &= \overline{B} \wedge \overline{(D \wedge (F \wedge G))} = \overline{B} \wedge (\overline{D} \vee \overline{(F \wedge G)}) \\ &= \overline{B} \wedge (\overline{D} \vee (\overline{F} \vee \overline{G}))\end{aligned} \tag{2}$$

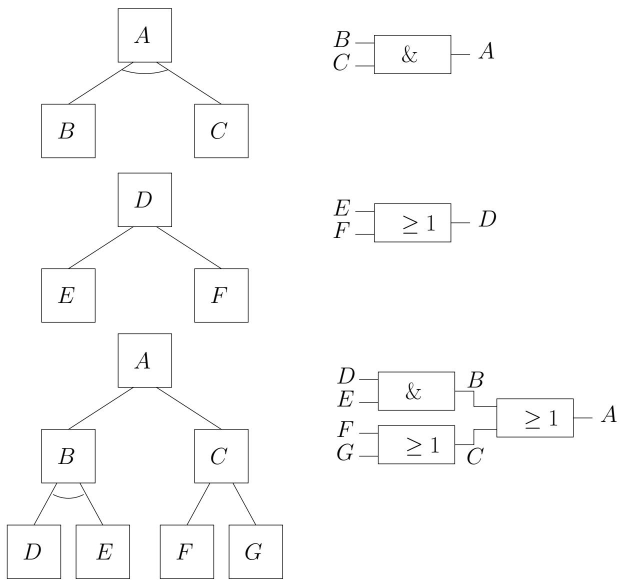

Fig. 3. Any given AtPT can be transformed into a logic network. Three examples (*top to bottom:* AND-connection, OR-connection, complex AtPT) are shown here using electric gate symbols.

The inverse of the logical representation of an AtPT represents all possible countermeasure scenarios, since an attack can only be accomplished if the attack associated with the root node is realized. The logical inverse, therefore, describes all combinations of realized/prevented sub-goals that prevent the root node and its attack from being achieved. Once the Boolean representation has been transformed back into a tree, referred to as a DePT, the connections between AtPT and DePT can be easily seen. Should the DePT contain ambiguous leaf nodes where a range of countermeasures could be selected, this is due to a badly defined AtPT, where the atomic attacks could be further refined by additional subdivisions.

Figure 4 contains the DePT for the taximeter example given in Figure 1. At every leaf node, the DePT automatically offers a possible countermeasure to prevent that atomic attack from being carried out: Node G, for instance (replacing an old seal with a forged new seal) can be countered by improving the quality of the applied seals. Other suggested countermeasures are less practical, however. To prevent new software from being installed (node F), a possible countermeasure is to make the procedure of installing software more difficult, e.g. by creating a non-standard programming interface. Unfortunately, such a measure would also affect the legal intended modification of software. Countermeasures to the remaining leaf nodes (B and D) should also be obvious. Node B (modification of parameters), can be prevented from happening by increasing parameter protection. Node D (writing new (fake) software for the taximeter) can be made more difficult by increasing the complexity of the API. To reiterate, this countermeasure would affect both attacker and original programmer of the software and it is, thus, impractical. Based on the DePT, it can be concluded that one way to prevent fare manipulation

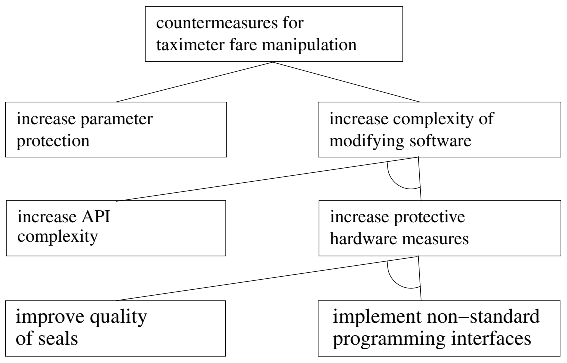
countermeasures for taximeter fare manipulation
increase parameter protection
increase complexity of modifying software
increase API complexity
increase protective hardware measures
improve quality of seals
implement non−standard programming interfaces

Fig. 4. DePT for the countermeasures needed to prevent the calculated fare/measurement value of a taximeter from being manipulated.

is implementing increased parameter protection measures and better physical seals. Other combinations of countermeasures are, of course, also possible. To select the optimal constellation of countermeasures (cheapest countermeasure set to lower the risk below a defined threshold) the attributes examined during the original risk assessment are needed.

B. Usage of Defense Probability Trees

As mentioned above, the simple Boolean view on AtPTs and DePTs lacks all information on attack attributes, probabilities of occurrence and subsequently risk. However, the DePT can be used in the same way as the AtPT to estimate defense costs and resistance to attacks: Here, it will be assumed that the impact of a realized threat is 1 (larger number of measurement results affected) and that a total risk of 3 or lower (probability score ≤ 3, sum score ≥ 14) is acceptable. Each leaf node in the DePT can then be assigned scores for time, expertise, knowledge, window of opportunity and equipment that represent the cost of implementing a countermeasure that reduces the risk of the corresponding leaf node in the AtPT to 3 or lower. The knowledge score now refers to the amount of research needed to realize a countermeasure. If all necessary information is easily available (for instance, because the defender is also the original developer) the knowledge would be considered readily available (score of 0). The window of opportunity then reflects the fact that a defender may not always have access to the attacked instrument. It should be noted that, just as for the attack scenario, not all countermeasures need to be implemented. Instead, the DePT will help to identify the set of countermeasures that produce an acceptable risk reduction at minimal cost/effort. The rules of attribute propagation within the DePT remain the same as listed in Section III-B, since the defender will try to minimize her efforts as well. The process of finding an optimal set of countermeasures can be summarized as follows:

1) Construct an AtPT according to the rules described in [5].
2) Derive a DePT from the resulting Boolean formulae and discard all branches that constitute impractical measures that affect both attacker and intended user.

3) Select implementations for each atomic countermeasure (leaf node in the DePT) that reduce the resulting risk for the specific node to 3 or lower and label the corresponding leaf node with the respective attribute scores.
4) Propagate these weights up the tree to identify the optimal (cheapest) set of countermeasures.

For the attribute scores for implementation of the countermeasure, the same values as for the implementation of attacks can be used, since they reflect the cost of implementation adequately. The only difference is the perspective of the party attempting an implementation of the countermeasure. For example, the window of opportunity for implementation of a countermeasure should almost always be easy (score of 0) since the user of the instrument will normally allow access to the instrument to close its vulnerabilities. A practical example that illustrates the four steps listed above is given in the following section.

V. Experimental Example: Assessment of the Meltdown vulnerability

The following assessment of the Meltdown vulnerability and its impact on measuring instruments was performed at PTB in late 2017. Manufacturers of affected instruments were afterwards contacted to close the vulnerability, when necessary. At the moment of writing this paper, all necessary software patches have been implemented and re-certified, preventing susceptible instruments to remain in the field.

A. Problem Description

The address space of all physical memory including the address space of the kernel of most operating systems such as Linux, Windows and macOS is mapped in the page table of each individual user process. To prevent processes from accessing certain addresses without permission, these are marked with a supervisory bit. If a process attempts to write data to or read data from such an address, an exception is triggered. However, between the actual access violation and the triggering of the exception, a certain time elapses during which the processor may have read data from the forbidden address and performed additional transient commands [7]. These actions affect the contents of the processor cache, which itself may be read out by the well established Flush-Reload or Evict-Reload algorithms [10]. Both algorithms depend on a precise time reference to determine the contents of the cache. Since this process can be repeated as often as needed, it is feasible to read out the entire contents of the RAM with a data rate of around 500kByte/s.
From a technical point of view, the Meltdown vulnerability can initially only be used to read data from the RAM without permission and thus violate the confidentiality of a specific piece of information. Confidentiality, however, does not belong to the security properties of the assets worthy of protection of a measuring instrument, which only address integrity, authenticity and availability. However, the vulnerability can be exploited to spy on secondary sensitive attributes, such as private keys used for asymmetric cryptographic signatures. For different types of measuring instruments, such private keys are used to cryptographically sign determined measurement results and thus to ensure that the measurement results can no longer be changed unnoticed after an export from the measuring instrument. In addition, the signature ensures that the results have been actually generated by the calibrated measuring instrument.
Within this example, only threats on the authenticity of the measurement data will be examined. The formal threat definition can therefore be stated as follows:

An attacker generates false measurement results.

B. Attack probability tree for the Meltdown scenario

The Attack Probability Tree (AtPT) shown in Figure 5 breaks down this threat into individual partial attacks / attack vectors, which must be combined to realize the threat. As before, attack vectors associated with an arc are read as AND-connected, all other attack vectors are OR-connected.

The attacks A01, A04, A07, A09, A10, A12, A13, and A14 represent atomic attack vectors that are no longer subdivided. From the AtPT shown in Figure 5, the following two basic requirements can be derived, which are necessary for exploiting a Meltdown attack on a measuring instrument to realize the threat:

- The measuring instrument's processor is affected by the Meltdown vulnerability.
- The measuring instrument has an open physical interface or a network connection that is protected by means of the operating system.

The attack is therefore unworkable if a hardware security module is used or if the measuring instrument has no open physical or logical interfaces.

C. Evaluation of atomic attack vectors

The score for atomic attack vectors according to the procedure described in [2] is shown in Table II. A brief description of all atomic attack vectors in order to execute the attack is presented below:

A13, A14: To introduce executable code into the instrument (A11), an attacker must use a physical interface (A13) or a network interface (A14).

A12: Once A11 has been achieved (attacker introduces executable code into the instrument), and the attacker is set to use a vulnerability of the operating system (A12), she has the ability to execute the Meltdown attack on the processor (A08).

If code is to be executed on the attacked measuring instrument with limited privileges, basically two variants are conceivable to differ, in terms of what detailed knowledge of the system architecture must be available to the attacker. The more probable scenario results from the lower sum score:

1) Known system architecture (known location of the key): Knowledge of the system: 11, Time: 1, Expertise: 3 (Total: 15)
2) Unknown system architecture (unknown location of the key):

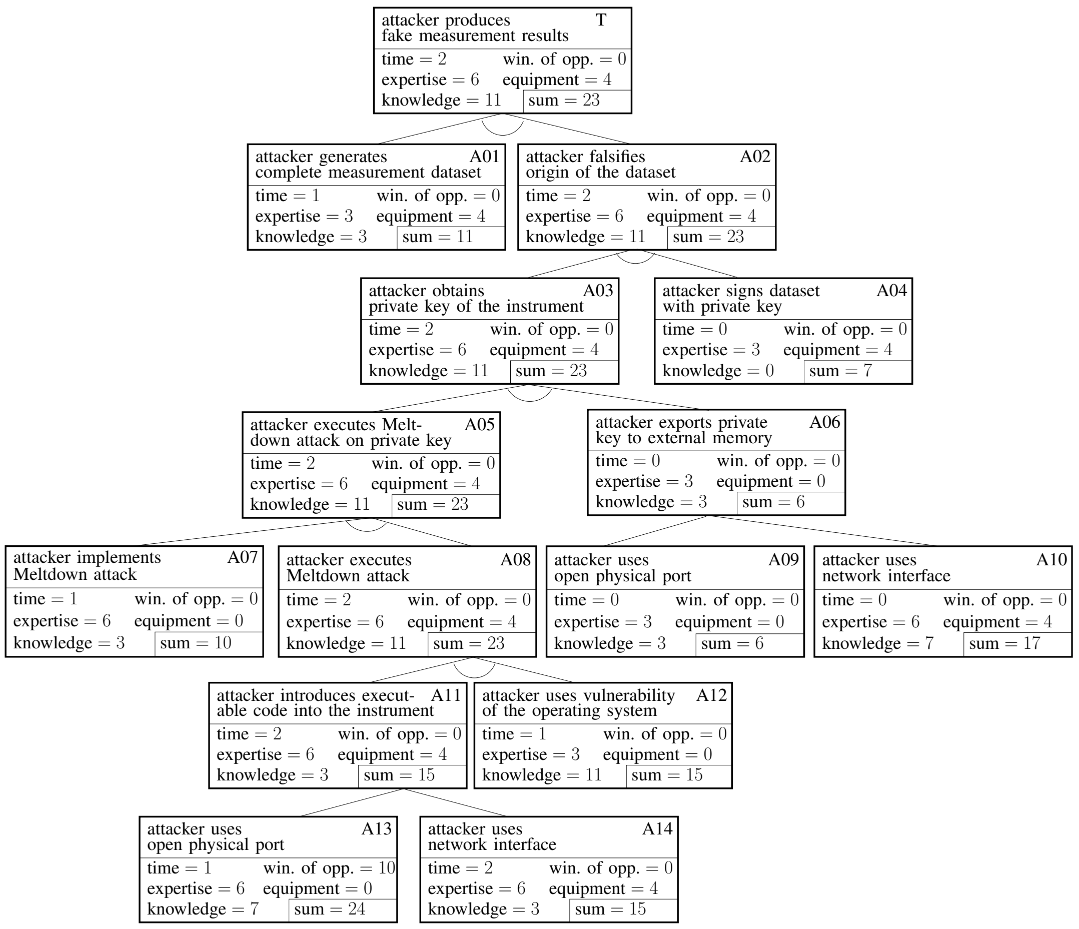
attacker produces fake measurement results T
time = 2 win. of opp. = 0 expertise = 6 equipment = 4 knowledge = 11 sum = 23
attacker generates complete measurement dataset A01
time = 1 win. of opp. = 0 expertise = 3 equipment = 4 knowledge = 3 sum = 11
attacker falsifies origin of the dataset A02
time = 2 win. of opp. = 0 expertise = 6 equipment = 4 knowledge = 11 sum = 23
attacker obtains private key of the instrument A03
time = 2 win. of opp. = 0 expertise = 6 equipment = 4 knowledge = 11 sum = 23
attacker signs dataset with private key A04
time = 0 win. of opp. = 0 expertise = 3 equipment = 4 knowledge = 0 sum = 7
attacker executes Melt- down attack on private key A05
time = 2 win. of opp. = 0 expertise = 6 equipment = 4 knowledge = 11 sum = 23
attacker exports private key to external memory A06
time = 0 win. of opp. = 0 expertise = 3 equipment = 0 knowledge = 3 sum = 6
attacker implements Meltdown attack A07
time = 1 win. of opp. = 0 expertise = 6 equipment = 0 knowledge = 3 sum = 10
attacker executes Meltdown attack A08
time = 2 win. of opp. = 0 expertise = 6 equipment = 4 knowledge = 11 sum = 23
attacker uses open physical port A09
time = 0 win. of opp. = 0 expertise = 3 equipment = 0 knowledge = 3 sum = 6
attacker uses network interface A10
time = 0 win. of opp. = 0 expertise = 6 equipment = 4 knowledge = 7 sum = 17
attacker introduces execut- able code into the instrument A11
time = 2 win. of opp. = 0 expertise = 6 equipment = 4 knowledge = 3 sum = 15
attacker uses vulnerability of the operating system A12
time = 1 win. of opp. = 0 expertise = 3 equipment = 0 knowledge = 11 sum = 15
attacker uses open physical port A13
time = 1 win. of opp. = 10 expertise = 6 equipment = 0 knowledge = 7 sum = 24
attacker uses network interface A14
time = 2 win. of opp. = 0 expertise = 6 equipment = 4 knowledge = 3 sum = 15

Fig. 5. Attack probability tree for generating false measurement results with a stolen private key by means of the Meltdown vulnerability. Attack vectors linked with an arc are AND-connected. All other attack vectors have an OR-connection. Note: The attack can only be implemented if the private key is available in the instrument at runtime and if the processor is affected by Meltdown.

Knowledge of the system: 7, Time: 4, Expertise: 6 (Total: 17)

A07: Once A08 has been achieved (attacker executes Meltdown attack), and the attacker has written the required code for spying out the private key, she can use the Meltdown attack on the private key (A05). When the attacker implements the actual attack to exploit the Meltdown vulnerability, she will probably create a full dump of all memory. Therefore, no special knowledge of the system is needed. Once again, the more probable attack scenario results from the lower sum score:

1) Attacker's own implementation:
   Knowledge of the system: 3, Time: 2, Expertise: 8 (Total: 13)
2) Attacker uses third parties' implementations:
   Knowledge of the system: 3, Time: 1, Expertise: 6 (Total: 10)

A09, A10: The writing or exporting of data is already provided in many measuring instruments, so that no increased effort has to be set for this. Either of these two subgoals allows exporting of the private key to external memory (A06). Once A06 has been achieved, and the attacker has executed the Meltdown attack on the private key (A05), the attacker can obtain the private key of the instrument (A03).

A04: Once the attacker has obtained the private key (A03), she is able to sign new datasets (A04); if A03 and A04 are achieved, the attacker can falsify the origin of fake datasets (A02).

A01: Finally, once the capacity for falsifying the origin of datasets is achieved (A02), and the attacker has generated a new (and fake) measurement dataset, the analysed Threat (T) can be realized.

In the case of an operating system configured in accordance with legal requirements, no code can be executed from an external medium. However, it cannot be ruled out that the Meltdown bug will not lead to speculative execution. It should be noted that attacks A01 (generation of a measurement dataset) and A04 (signing of a measurement dataset) must be repeated for each individual measurement result and therefore should be assigned a reduced impact of $\frac{1}{3}$. However, since these are attacks with comparatively little effort, the overall impact rating does not change. Once the rules for attribute

TABLE II
EVALUATION OF ATOMIC ATTACK VECTORS - THE ATTACK VECTORS A01 AND A04 DIFFER FROM ALL OTHERS IN THAT THEY MUST BE REPEATED FOR EACH INDIVIDUAL REALIZATION OF THE THREAT. ACCORDINGLY, THE DAMAGE HERE IS REDUCED TO A VALUE OF 1/3.

Attack ID	attack vector	time	expertise	knowledge	window of opp.	equipment	sum	impact
A13	Attacker uses open interface to bring code into instrument.	1	6	7	10	0	24	1
A14	Attacker uses network connection to bring code into instrument.	2	6	3	0	4	15	1
A12	Attacker expoits operating system vulnerabilites to execute code with limited privileges.	1	3	11	0	0	15	1
A07	Attacker implements Meltdown attack.	1	6	3	0	0	10	1
A09	Attacker uses open interface to export private key.	0	3	3	0	0	6	1
A10	Attacker uses network connection to export private key.	0	6	7	0	4	17	1
A04	Attacker signs measurement result with private key.	0	3	0	0	4	7	$\frac{1}{3}$
A01	Attacker generates complete measurement dataset.	1	3	3	0	4	11	$\frac{1}{3}$

propagation have been applied, the root node has a sum score of 23, which is equivalent to a probability score of 2 and a risk score of 2. Such a rating would be acceptable for most instrument classes except when the law requires even more stringent protective measures, in which case the risk has to be reduced to 1.

D. Identification and Selection of Suitable Countermeasures

As described in Section IV the DePT (see Figure 6) can be inferred by converting the AtPT to a Boolean expression and applying the inverse operation on the expression. The resulting graphical representation is given in Figure 6. It should be noted that the structure of the tree has been modified according to the transformation rules laid out in [6] for better comprehensibility. From this initial DePT all nodes can be removed that constitute impractical defensive measures, i.e. making the signature algorihm more complex (node D04), deactivating all hardware and software interfaces (node D11/D13 and D12/D14) since all three are needed for the actual intended operation of the instrument. Afterwards, a number of nodes remain that have only one child node. According to [6], such a parent node can be replaced by the child node since its attributes must be identical to those of the child node. After the removal of all impractical defensive measures, the DePT is reduced to a tree of three nodes, see Figure 7. The remaining DePT states that retrieving the private

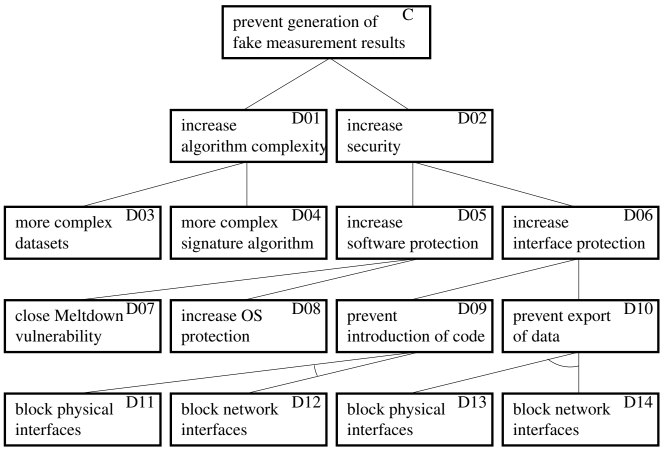
prevent generation of fake measurement results C
increase algorithm complexity D01
increase security D02
more complex datasets D03
more complex signature algorithm D04
increase software protection D05
increase interface protection D06
close Meltdown vulnerability D07
increase OS protection D08
prevent introduction of code D09
prevent export of data D10
block physical interfaces D11
block network interfaces D12
block physical interfaces D13
block network interfaces D14

Fig. 6. Defense probability tree for generating false measurement results with a stolen private key by means of the Meltdown vulnerability. Defense vectors linked with an arc are AND-connected. All other defense vectors have an OR-connection.

key from within a measuring instrument via exploitation of the Meltdown vulnerability to generate false measurement data can be prevented by one of two alternatives:

- external code is prevented from being executed through increased protective operating system measures (node D08),
- a patch is applied to the operating system to prevent speculative code execution, thus closing the vulnerability (node D07).

When evaluating these alternatives, all assigned scores (time, expertise, knowledge, window of opportunity and equipment) have to be based on the assumption that the implementation is done by a white hat developer who has access to both the instrument, as well as all manufacturer's documentation.

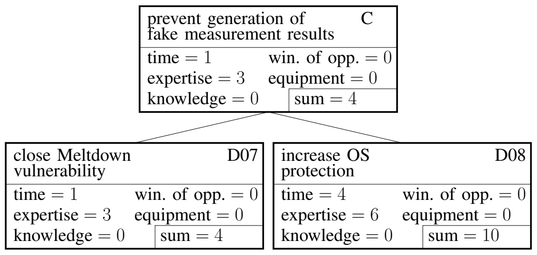
prevent generation of fake measurement results C
time = 1 win. of opp. = 0
expertise = 3 equipment = 0
knowledge = 0 sum = 4
close Meltdown vulnerability D07
time = 1 win. of opp. = 0
expertise = 3 equipment = 0
knowledge = 0 sum = 4
increase OS protection D08
time = 4 win. of opp. = 0
expertise = 6 equipment = 0
knowledge = 0 sum = 10

Fig. 7. Reduced DePT for the Meltdown vulnerability. Nodes with only one child node have been replaced by the child node in accordance with the rules laid down in [5].

Concerning the implementation of increased protective operating system measures (D08), a programming expert (score of 6 for expertise) should be able to implement, test and release a solution within a month (score of 4 for time). Since all internal manufactuer's documentation is available to the white hat developer, the knowledge score shall be set to 0 (readily available). The same is true for the window of opportunity. Furthermore, no special equipment will be needed to implement a software solution. Similarly, applying an operating system patch to close the vulnerability (D07) should be quite easy.

Since many operating system manufacturers provide patches and instructions for their installation on their website, any proficient user (expertise score of 3) should be able to patch the operating system within a week. All necessary information should be readily available (knowledge score of 0), while access to the instrument is always granted. Also, as was the case for node D08, only standard equipment is needed for the implementation of the countermeasure.

E. Conclusion of Experimental Example

Once the two leaf nodes of the DePT have been labeled with these attributes, see Figure 7, the rules laid down in Section III-B are applied to yield the attributes for the root node. This results in the root node becoming a copy of leaf node D07, which indicates that application of an operating system patch is the easiest countermeasure against the defined threat. Once the Meltdown vulnerability has been closed as described, the only way to get the private key of the instrument (node A03 in Figure 5) is to correctly calculate the key by a brute-force method, taking longer than half a year for any state-of-the-art signature algorithm. Since this will set the time score for node A03 to 19 points, the root node will be similarly affected, increasing its sum score to 40 (equivalent to a probability score and risk score of 1). Therefore, the proposed countermeasure is suitable to prevent the examined threat.

The same procedure can be applied to other threats with larger DePTs. The logic behind choosing a countermeasure should be, in general, identical to the logic of attack implementations, since both attacker and defender will aim to achieve their goals with minimal effort. Therefore, all previously performed evaluations of AtPTs will hold true also for DePTs. To validate the usefulness and efficiency of the proposed countermeasure identification and selection method, more exemplary applications are needed, of course. Nevertheless, this investigation should be seen as a first proof of concept. Within the scope of this paper, the action associated with a node has either been assumed to have a permanent effect (impact score of 1) or its influence on the overall cost of implementing an attack/countermeasure was assumed to be negligible. Since this assumption will eventually fail in certain scenarios, an approach to deal with differing impact scores for individual nodes is still needed.

VI. Summary

Evaluating IT threats to assets and selecting appropriate countermeasures will form a cornerstone of Legal Metrology in the near future. While other IT sectors are already using such risk-based evaluation schemes, the present paper describes a suitable method for risk mitigation tailored for measuring instruments subject to legal control. To this end, AtPTs with calculation rules for their attributes have been transformed into equivalent DePTs. Since the method is based on established international standards, it is anticipated that it can easily be applied in other economic sectors as well. An application of the method [5] on the Meltdown vulnerability has been used to demonstrate the workflow and usage of the presented countermeasures selection procedure. In the future, the impact of vulnerabilities in common IT products on Legal Metrology will certainly need to be investigated more frequently. The risk analysis and countermeasure derivation methods discussed in this paper represent a selection of tools, at the disposal of notified bodies, manufacturers and market surveillance authorities, to asses the risks associated with such IT-related incidents. One piece that is still missing for AtPTs and DePTs to be generally applicable is a method to deal with attacks/countermeasures with different impact scores (i.e. permanent and repetitive) within one specific AtPT/DePT alike. Further work will address a theoretical analysis of the influence of impact scores on the overall risk and a proposal to reflect such scores in AtPTs and DePTs. Finally, a harmonized guideline for using AtPTs and DePTs should be written to be applied to new technologies, to test the efficiency of both methods for streamlining innovations within Legal Metrology.

References

[1] EC, "Directive 2014/32/EU of the European Parliament and of the Council of 26 February 2014 on the harmonisation of the laws of the Member States relating to the making available on the market of measuring instruments," European Union, Council of the European Union ; European Parliament, Directive, February 2014.

[2] M. Esche and F. Thiel, "Software risk assessment for measuring instruments in legal metrology," in *Proceedings of the Federated Conference on Computer Science and Information Systems*, Lodz, Poland, September 2015. doi: http://dx.doi.org/10.15439/978-83-60810-66-8 pp. 1113–1123.

[3] ISO/IEC, "ISO/IEC 27005:2011(e) Information technology - Security techniques - Information security risk management," International Organization for Standardization, Geneva, CH, Standard, June 2011.

[4] ——, "ISO/IEC 18045:2008 Common Methodology for Information Technology Security Evaluation," International Organization for Standardization, Geneva, CH, Standard, September 2008, Version 3.1 Revision 4.

[5] M. Esche, F. Grasso Toro, and F. Thiel, "Representation of attacker motivation in software risk assessment using attack probability trees," in *Proceedings of the Federated Conference on Computer Science and Information Systems*, Prague, Czech Republic, September 2017. doi: http://dx.doi.org/10.15439/2017F112 pp. 763–771.

[6] S. Mauw and M. Oostdijk, "Foundations of attack trees," in *Proceedings of the 8th international conference on Information Security and Cryptology*. Seoul, Korea: IEEE, December 2005. doi: http://dx.doi.org/10.1007/11734727_17 pp. 186–198.

[7] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018.*, 2018, pp. 973–990. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/lipp

[8] P. Wang, W.-H. Lin, P.-T. Kuo, H.-T. Lin, and T. C. Wang, "Threat risk analysis for cloud security based on attack-defense trees," in *Proceedings of the International Conference on Computing Technology and Information Management*. Seoul, Korea: IEEE, April 2012, pp. 106–111, ISBN: 978-89-88678-68-8.

[9] R. Vigo, F. Nielson, and H. R. Nielson, "Automated generation of attack trees," in *Proceedings of the IEEE Computer Security Foundations Symposium*. Seoul, Korea: IEEE, 2014. doi: http://dx.doi.org/10.1109/CSF.2014.31 pp. 337–350.

[10] Y. Yarom and K. Falkner, "FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack," in *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.*, 2014, pp. 719–732. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/yarom