Athens Journal of Technology and Engineering - Volume 6, Issue 4 – Pages 223-238

Using an Extended Attack Defense Graph Model to Estimate the Risk of a Successful Attack on an IT Infrastructure

By Christoph Karg[*] *& Till Hänisch*[†]

Nowadays, securing the IT infrastructure is an ongoing task in every company and organization. For small and medium-sized enterprises, this task is challenging because of its complexity and the related costs. Especially the risk assessment of threats and the choice of appropriate countermeasures is hard to handle by this kind of enterprises. Using the example of a ransomware attack, this paper describes how to use a method for risk assessment on the basis of attack defence graphs and Monte Carlo simulations. The details of the simulation algorithm are explained and formal aspects are considered.

Keywords: *IT security, attack defense graph, risk assessment, threat modeling.*

Introduction

In a commercial environment, the goal of each security measure is the protection of the company's or organization's assets. In the context of information security, the measures focus on computer systems and the data stored on them. Usually, the financial budget to be spent on security measures is limited. As a consequence, not all of the available measures can be applied because of monetary restrictions. Hence, a choice must be made on how much money to spend on which security measure in order to use the financial resources in an optimal manner.

The selection of the measures to secure a company's IT infrastructure is usually based on best practices. In the case of office IT environments, many of the choices are based on the experience which was gathered in the last decades. In the case of industrial production environments, the situation is quite different, since the process of digitalization just begins to find its way into these environments. According to IT security experts, there is the need of a quantitative assessment of a security measure with respect to the environment it shall be applied to (Blakley et al. 2002, Cremonini and Martini 2005). This kind of assessment may assist decision makers in choosing and prioritizing appropriate security measures.

This paper describes an approach to assess the effectiveness of security measures on the basis of Monte Carlo simulations. The approach builds on the well-known model of attack defense graphs. An attack defense graph is a directed acyclic graph whose nodes represent threats which arise from existing vulnerabilities and countermeasures to mitigate the respective threats. The nodes are grouped in compositions (and) or alternatives (or) in order to specify the dependencies. Each sink of the graph represents an attack which may be the result

[*]Professor, Aalen University of Applied Sciences, Germany.
[†]Professor, DHBW Heidenheim, Germany.

https://doi.org/10.30958/ajte.6-4-2 doi=10.30958/ajte.6-4-2

of successfully exploiting the vulnerabilities which are located on the paths towards the sink. The attack may be prevented by successfully applying the counter measures which lie on the paths towards the sink.

In order to estimate the risk of a successful attack, the model is extended with additional information. In particular, both a capability and a difficulty value are assigned to each node representing a threat or a countermeasure. The capability of a threat or a countermeasure describes the skill level of an attacker to successfully implement the threat or the skill level of a defender to successfully deploy a countermeasure, respectively. It is assumed that the capability value is independent of the environment to be analyzed. The difficulty value of a node measures the difficulty of implementing a threat or of deploying a countermeasure in a given environment, respectively. To enable Monte Carlo simulation techniques, probabilities are derived from the capability and the difficulty values.

To answer questions such as "How does the usage of security measure A influence the risk?" or "Is security measure A better than security measure B with respect to the mitigation of the risk?“, several Monte Carlo simulations are performed and analyzed. The simulation results can help the decision makers to select these countermeasures which fit the best to their IT environment. Another application of the approach is the computation of an cost-optimal selection of security measures which minimize the risk of a successful attack. The use case of a ransomware attack is used in order to illustrate the application of the model.

The paper is organized as follows. The section *Related Work* contains a summary of the papers which were relevant for this work. The section *Findings/ Results* contains our contributions to the topic. At first, the model of attack defense graphs is introduced briefly. Then the model is applied to the use case of a ransomware attack. After describing the algorithm behind the simulation system, the use case is analyzed. Furthermore, formal aspects of the model are presented and insights into the implementation are provided. The paper closes with the section *Conclusion*.

Related Work

Analyzing the safety of a technological system with the mathematical model of graphs is a well-known and acknowledged methodology in systems engineering. The roots go back in the 1960s where Watson and Mears at Bell Labs developed a tree-based technique to analyze the Minuteman Launch Control System. Hassl from Boeing recognized the potential of this approach and promoted it as a significant system safety analysis tool. The tool became popular as fault tree analysis (FTA) in the aerospace industry and was adopted from other industries such as the nuclear power industry and the robotics industry. In 1981, the U.S. Regularity Commission published a handbook on the application of the fault tree analysis and its mathematical foundations (Vesely et al 1981). Over the last six decades, fault tree analysis was developed further by a worldwide scientific community. More details on the history of the fault tree analysis can be found in

(Ericson, 1999). An approach to utilize attack-fault trees for quantitative analysis in the area of cyber physical systems is given in (Kumar and Stoelinga 2017).

In the year 1999, Bruce Schneier introduced the concept of attack trees as a (Schneier 1999a, 1999b). According to Schneier, most people do not have a detailed understanding in computer security. Especially, for decision makers in companies and organizations it is hard to figure out the consequences of a cyber security threat. Attack trees are a formal method to describe attacks on computer systems in a manner which is understandable for non-experts. Schneier proposed to assign attributes to the nodes in order to enrich the attack tree with additional information. Examples of such attributes are the success possibility of the attack represented by an node, the equipment needed to perform the attack, or the costs of the attack to be paid by the threat agent. Schneier's model of attack trees is a simplistic one and lacks the concepts of countermeasure nodes and success probabilities.

Schneier's concept influenced the work of many researchers on the field of computer security. For example, Mauw and Oostdijk (2005) studied formal aspects of attack trees and provided a denotional semantics. To do this, they formalized the notion of an attack tree and studied transformations on attack trees and their respective consequences (Mauw & Oostdijk, 2005). Kordy et al. (2014) extended the model to so-called attack defense trees by adding countermeasures to the tree (Kordy et al. 2014). The idea behind their approach is to model a game between an attacker and a defender of a computer system. On the one hand, the attacker has the goal to successfully realize the threat by applying the steps represented by the attack nodes. On the other hand, the defender tries to prevent the attacker from being successful by applying the countermeasures described in the defense nodes. The authors gave a formal representation of the attack defense tree model and prove several semantic aspects of the model.

Edge et al. (2006) used attack and protections trees to analyze attacks against computer networks (Edge et al. 2006). They created the concept of threat logic trees. These are trees where metrics are associated to the leaf nodes of the tree. The metrics are probability of success, impact to the system, the cost to attack, and risk. The metrics are used to analyze the tree and to estimate the risk of a successful attack. They use their model to analyze a distributed denial-of-service attack to the servers of Homeland Security. In contrast to the above approaches, the modelling is split in two trees: the attack tree and the protection tree.

The usage of attack defense trees in threat modelling and risk assessment is a widely recognized methodology in information technology. For example, Fraile et al (2016) used attack defense trees to analyze the security of automated teller machines (ATM) (Fraile et al. 2016). Based on their practical work, the authors attest attack defense trees a high potential to produce good results in risk assessment.

The quality of the modelling with attack defense trees is strongly related to the experience of the persons which are involved in the analysis process. Experts on the field of cyber security apply methodologies from risk assessment such as the OWASP risk rating methodology (OWASP 2019) or the CORAS approach (Lund et al. 2011). Another useful tool is the common vulnerability scoring system

(CVSS) which is a worldwide accepted standard to describe the characteristics of vulnerability (Forum of Incident Response and Security Teams 2019). CVSS is used to classify known vulnerabilities in a standardized way. The findings are published in publicly available databases such as the U.S. National Vulnerability Database[1].

Hänisch and Karg (2019) introduced a method for cyber security risk assessment on the basis of attack defense graphs and Monte Carlo simulations (Hänisch and Karg 2019). Their goal was to make the importance of specific measures transparent to decision makers and to support security specialists without requiring the effort of a complete risk analysis.

Findings & Results

The Model

The model to be used in following is a combination of attack defense graphs and Monte Carlo simulations. For a detailed introduction to the model, we refer to (Hänisch and Karg 2019).

In this model, attack defense graphs are used to analyze the risk of a threat. Such a graph is a directed acyclic graph. The nodes of the graph consist of threats, countermeasures, or selections of them. A selection can be either a composition (and) or an alternative (or). The root of the graph represents the main threat to be analyzed. The leafs of the graph are the atomic actions to be performed by the attacker and the defender, respectively. Each leaf is assessed with two values, the *capability* and the *difficulty*. The meaning of the values is as follows:

- The capability measures the fundamental complexity of successfully applying the threat or the countermeasure. The value is integral and ranges from 1 (simple) to 6 (very complex).
- The difficulty rates the environment the system is located and the respective effects on implementing the threats and countermeasures. The value is integral and ranges from 1 (simple) to 6 (very complex).

The success rate of a threat or a countermeasure is derived by its capability and difficulty assessment according to (Table 1). In the following, we refer to the contents of this table as the risk assessment model.

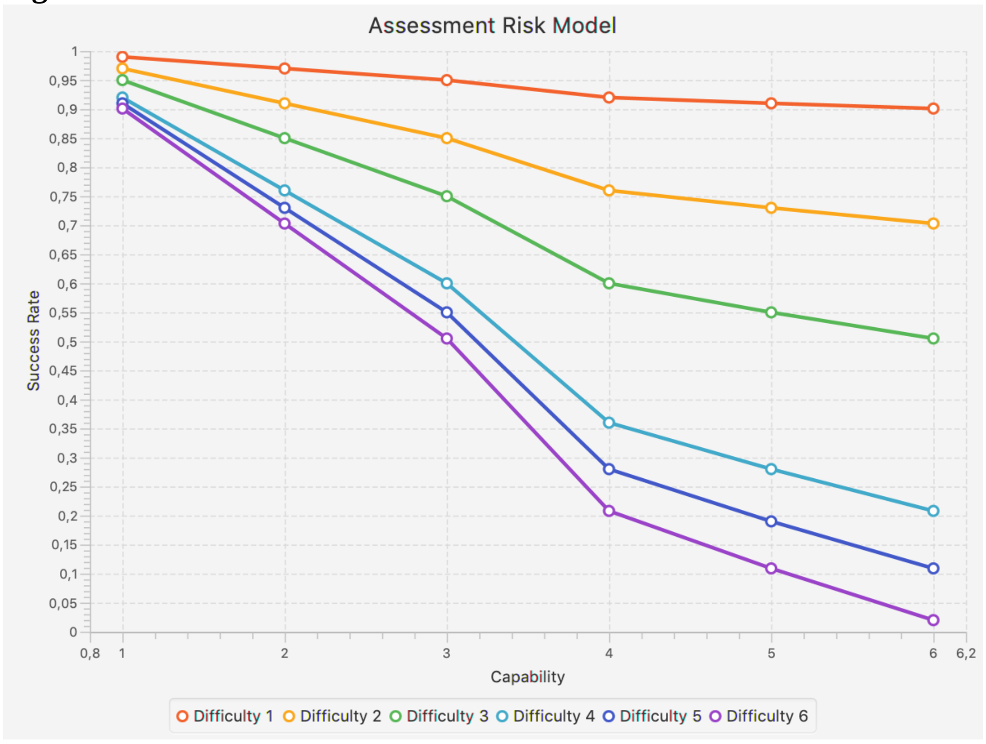
[1]Webpage: https://nvd.nist.gov

Table 1. *Risk Assessment Lookup Table*

		capability					
		1	2	3	4	5	6
difficulty	1	0.99	0.97	0.95	0.92	0.91	0.90
	2	0.97	0.91	0.85	0.76	0.73	0.70
	3	0.95	0.85	0.75	0.60	0.55	0.51
	4	0.92	0.76	0.60	0.36	0.28	0.21
	5	0.91	0.73	0.55	0.28	0.19	0.11
	6	0.90	0.70	0.51	0.21	0.11	0.02

Figure 1 provides a chart of the success rates of the capabilities with respect to the difficulties.

Figure 1. *Risk Assessment Chart*

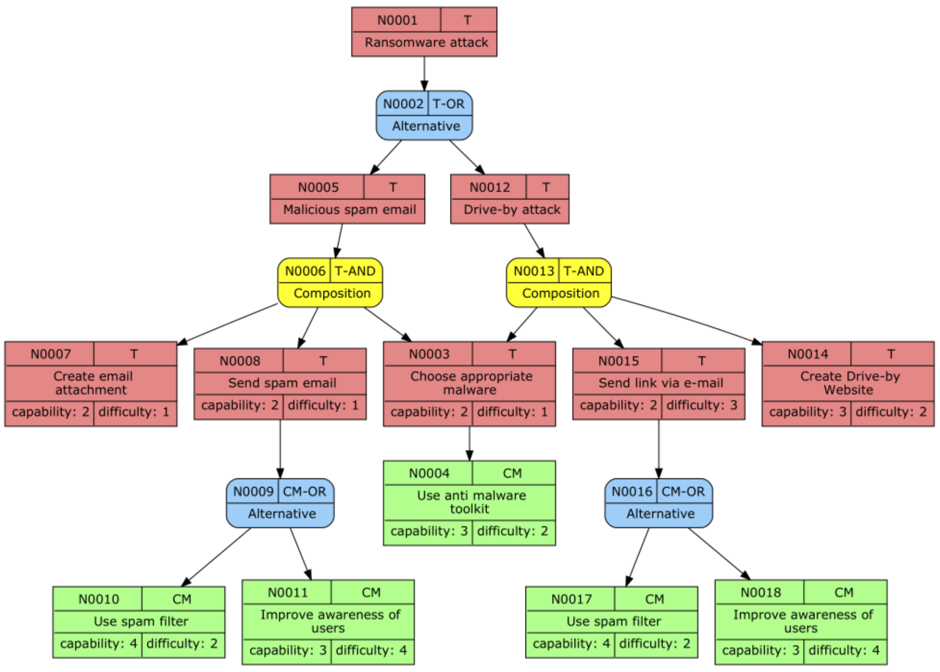


Use Case: Ransomware Attack

In the following, the attack defense graph model is used to analyze the threat of a ransomware attack. The respective graph is displayed in Figure 2.

The root (N0001) of the graph represents the risk of a successful ransomware attack. An assumption in this use case is that the attacker is not able to get physical access to the company site. Hence, he cannot place malicious USB sticks or hack the company's computers directly.

Figure 2. *An Attack Defense Graph modeling the Threat of a Ransomware Attack and the Respective Countermeasures.*



Without direct access, the attacker needs to the malware via the internet. He has two alternatives: sending a spam email which contains the malware as an attachment or performing a drive-by attack where the victim downloads the malware from a well-prepared web page. This alternative is modeled with the nodes N0002, N0005, and N0012.

Independently of the attack path, the attacker needs to choose a ransomware malware which is feasible for a successful attack (node N0003). This can be done quite easily by searching the internet or the darknet. A countermeasure is the deployment of an anti-malware toolkit within the company's IT infrastructure (node N0004). The challenge is to keep this toolkit up to date in order to detect the latest malware.

The next step of the attack path is the creation of an attachment containing the malware (node N0007). Usually, this is a PDF or a Microsoft Office document. From the defender's point of view, this cannot be prevented. Finally, the spam email needs to be sent (node N0008). The attacker can choose a service offered in the darknet or can use a public email service provider. The defender can act against this threat by using a spam filter (node N0010) or improving the awareness of the employees (node N0011).

After choosing the malware, the attack path continues with the setup of a website which is used to distribute the malware (node N0014). In the internet, there exist various hosting platforms for this purpose. It is not difficult to create a well-designed website. Finally, the attacker needs to send the link to the website via email (node N0015). The defender can take care on this threat by using a spam filter (node N0017) or improving the awareness of its employees (node N0018). In contrast to the previous attack path, we assume that for a spam filter it is more difficult to detect malicious links in an email than to detect bad attachments. This results in an higher capability value of node N0017.

Monte Carlo Simulation

After modeling the use case, the resulting attack defense graph is analyzed by performing a Monte Carlo simulation. The algorithm behind this simulation is depicted in Figure 3.

In the first step of the algorithm, the success rates of the leafs are initialized with the values of the risk assessment model. Then, the parameters of the simulation are initialized. All in all $\mathtt{total} = 100000$ simulation steps are performed. The number of successful simulation steps is stored in the variable cntr.

A simulation step is computed by a recursive procedure which performs a depth-first walk through the attack defense graph beginning at the root node. The steps of the computation depend of the type of visited node.

If the visited node is a countermeasure node, then the procedure chooses the success rate p depending on the countermeasure's capability and the difficulty (see (Table 1)) and determines its success by a random trial, this is, a Bernoulli experiment with success probability p.

If the visited node is a threat node, there are two cases:

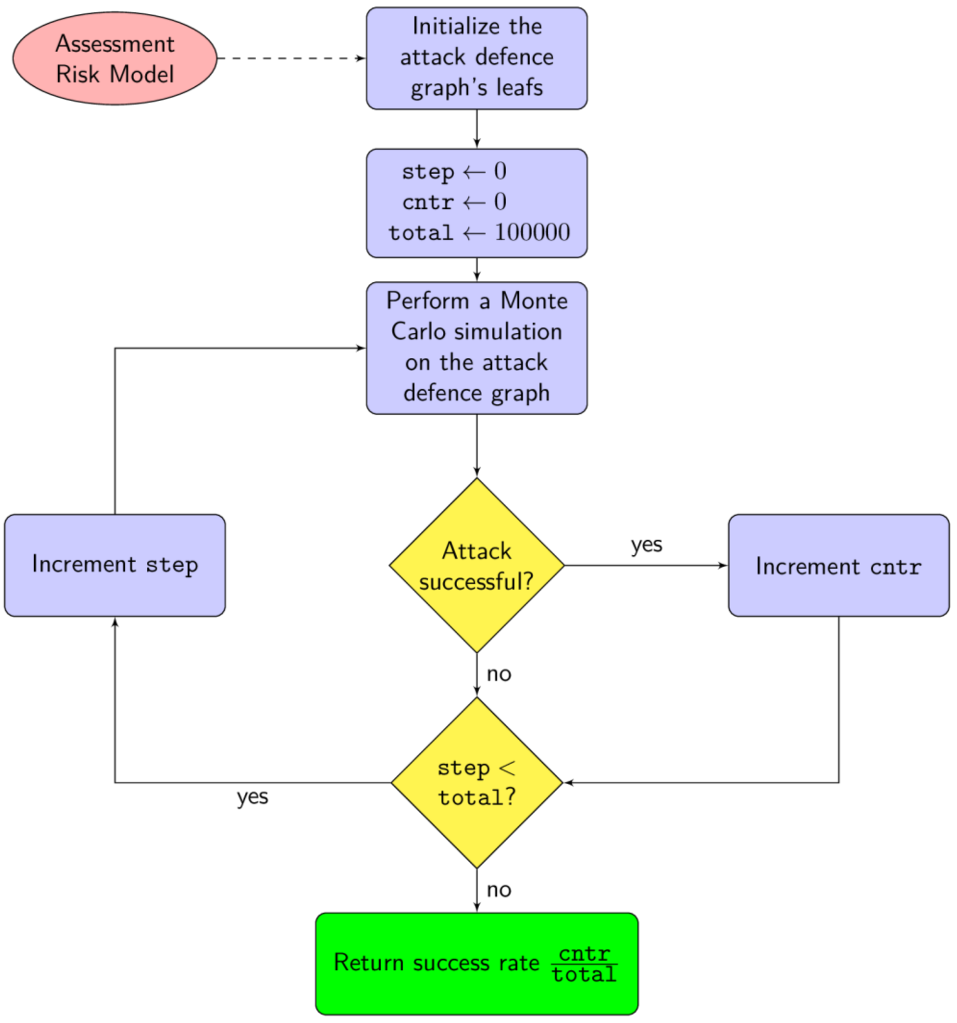
- The node is a leaf: In this case, the procedure guesses the result b_T of the threat uniformly at random according to its capability and difficulty.
- The node has a child threat: In this case, the procedure computes the result of the child threat recursively and stores it as the result b_T of the node.

If a countermeasure is assigned to the threat node, then the procedure furthermore computes the result b_C recursively and sets the result of the node to $b_T = b_T \ and \ not \ b_C$. This is, if the countermeasure is successful, then it prevents the success of the threat.

If the node is a composition (and) or an alternative (or) of threats or countermeasures, then the procedure recursively computes the result of the node's children and then sets the value of the node as the logical and or the logical or of its children, respectively.

In the case of a successful attack, the variable cntr is increased. As the result, the algorithm returns the ratio of successful attacks with respect to the total number of simulations, this is, the fraction $\frac{\mathtt{cntr}}{\mathtt{total}}$.

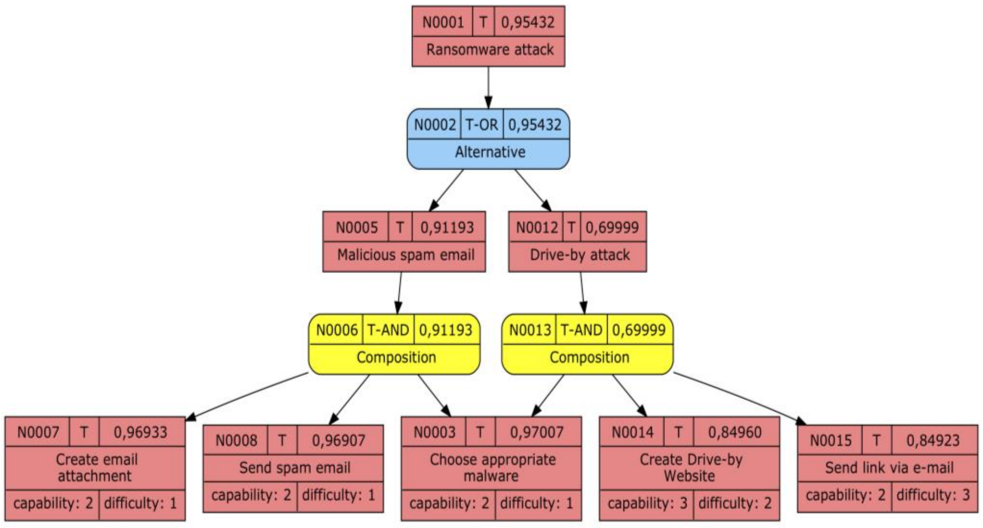
Figure 3. *Simulation Approach*


Assessment Risk Model
Initialize the attack defence graph's leafs
step ← 0
cntr ← 0
total ← 100000
Perform a Monte Carlo simulation on the attack defence graph
Attack successful?
yes
Increment cntr
no
Increment step
step < total?
yes
no
Return success rate cntr/total

Analysis of the use Case

The analysis of the use case starts with two basic simulations. The first simulation runs on a variant of the attack defense graph where all countermeasures are disabled (see Figure 4). According to the simulation, the success rate of threat without countermeasures is approximately 95.432%. The success rate is too high to be neglected.

Figure 4. *The Simulation of the use Case without Countermeasures Delivers a Success Rate of 95.432%.*

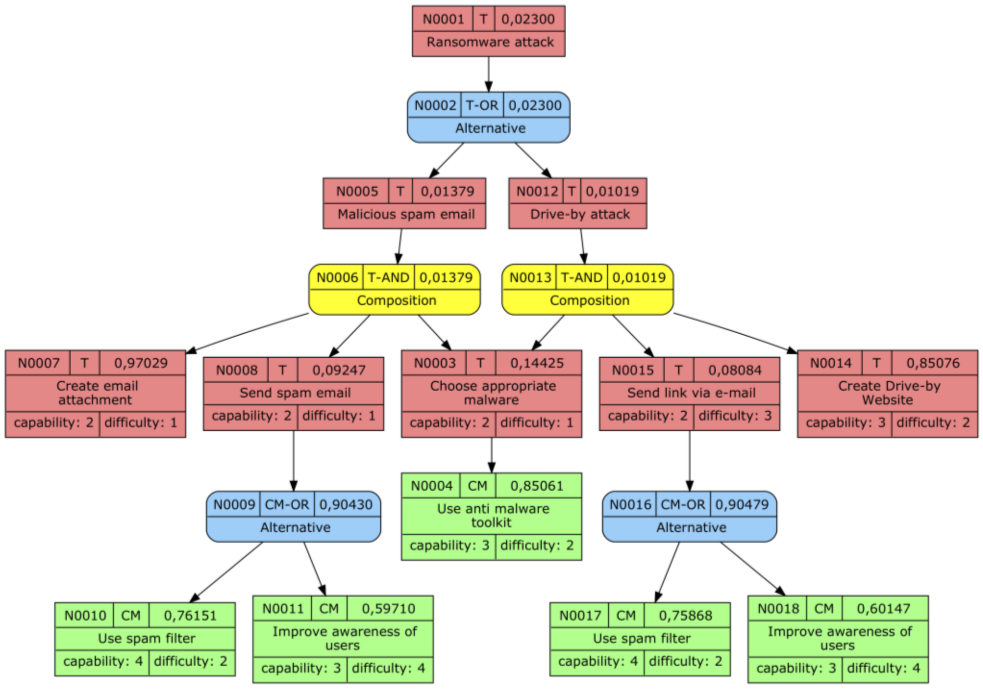


The second simulation runs on the graph with all countermeasures enabled (Figure 5). The result is that the countermeasures reduce the success rate to 2.3%. Interpretation: the risk of a ransomware attack can be effectively minimized by applying all countermeasures.

Table 2. *Sensitivity Analysis of the Usage of an anti-Malware Toolkit (node N0004)*

capability	difficulty	success rate
6	6	0.14551
6	5	0.13320
5	6	0.13051
5	5	0.11903
6	4	0.11720
4	6	0.11690
5	4	0.10623
4	5	0.10619
4	4	0.09512
3	6	0.07384
6	3	0.07324
5	3	0.06768
3	5	0.06725
4	3	0.05908
3	4	0.05903
6	2	0.04496
5	2	0.03836
3	3	0.03646
4	2	0.03437
3	2	0.02187

Figure 5. *Applying the Countermeasures Lowers the Success Rate to 2.3%.*



The usage of an anti-malware toolkit (node N0004) is a central countermeasure in the sense that it influences both attack paths. Hence, it is important to analyze the consequences of a wrong assessment of this node. This is done by changing both the capability and the difficulty of the node and performing another Monte Carlo simulation on the modified attack defense graph. The result is displayed in [tab: anti-malware-sensitivity]. The simulation shows that in the worst case the success rate of the ransomware attack is seven times higher (capability 6, difficulty 6) than in the initial assessment.

From an economic point of view, the deployment of IT security mechanisms results in costs such as license fees or working time of the IT department. A legitimate question is which of the counter measures can be omitted without significantly increasing the risk of a successful attack. These kind of questions can be answered by modifying the attack defense graph. In the use case of the ransomware attack, the implementation of the countermeasures (nodes N0011 and N0018) might cost a non-negligible amount of money. What happens if these countermeasures are omitted? Changing the attack defense graph and performing a Monte Carlo simulation shows that this results in a success rate of 5.233%.

Formal Aspects

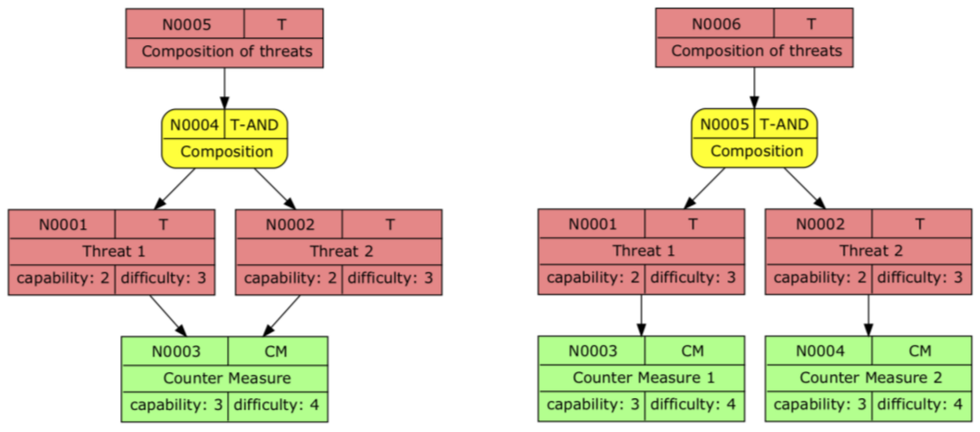
A fundamental assumption of the model is the independence of the leaf nodes. Using this assumption and a pocket full of mathematics, several facts can be derived which help to understand the behaviour of the model. These facts help to find errors in the implementation of the simulation system.

In the following T_1 and T_2 denote the event that the threat T_1 or the threat T_2 is successful, respectively. The events C, C_1, and C_2 are defined analogously with respect to countermeasures.

Fact 1. If T_1 and T_2 are independent, then
$\Pr[T_1 \cap T_2] = \Pr[T_1] \cdot \Pr[T_2]$,
and
$\Pr[T_1 \cup T_2] = 1 - \Pr[\overline{T_1}] \cdot \Pr[\overline{T_2}]$.

The first equation represents a composition of threats, the second one represents an alternative of threats. The fact applies to countermeasures too and can be extended to any number of independent events.

Figure 6. *Two ways to apply Countermeasures to a Composition of Threats*



a) One countermeasure for both threats b) one countermeasure per threat

A common situation during the modeling phase of a threat assessment is choice of preventing two threats together with one single countermeasure or with one countermeasure per threat (see (Figure 6)). The following two facts model the situation in the case of a composition (and) of two threats. Fact 2 addresses the first case ((Figure 6) a), fact 3 addresses the second case ((Figure 6) b).

Fact 2. If the events T_1, T_2 and C are independent, then
$\Pr[(T_1 \cap T_2) \cap \overline{C}] = \Pr[T_1] \cdot \Pr[T_2] \cdot \Pr[\overline{C}]$.

Fact 3. If the events T_1, T_2, C_1 and C_2 are independent, then
$\Pr[(T_1 \cap \overline{C_1}) \cap (T_2 \cap \overline{C_2})] = \Pr[T_1] \cdot \Pr[\overline{C_1}] \cdot \Pr[T_2] \cdot \Pr[\overline{C_2}]$.

If $\Pr[C] = \Pr[C_1] = \Pr[C_2]$, then the following inequality can be derived from fact 2 and fact 3:

$$\Pr[(T_1 \cap \overline{C}) \cap (T_2 \cap \overline{C})] < \Pr[(T_1 \cap \overline{C_1}) \cap (T_2 \cap \overline{C_2})].$$

This inequality can be interpreted as follows: in the case of a composition of two threats, it is better to mitigate the risk of each threat with a separate countermeasure than to handle both threats together with one single countermeasure.

The next two facts consider the two cases in the case of an alternative (or). Fact 4 addresses the usage of one countermeasure for both threats, fact 5 addresses the treatment of each threat with one separate countermeasure.

Fact 4. If the events T_1, T_2 and C are independent, then
$\Pr[(T_1 \cup T_2) \cap \overline{C}] = (1 - \Pr[\overline{T_1}] \cdot \Pr[\overline{T_2}]) \cdot \Pr[\overline{C}]$.

Fact 5. If the events T_1, T_2, C_1 and C_2 are independent, then
$\Pr[(T_1 \cap \overline{C_1}) \cup (T_2 \cap \overline{C_2})] = 1 - (1 - \Pr[T_1] \cdot \Pr[\overline{C_1}])(1 - \Pr[T_2] \cdot \Pr[\overline{C_2}])$.

If $\Pr[C] = \Pr[C_1] = \Pr[C_2]$, then the following inequality can be derived from fact 4 and fact 5:

$$\Pr[(T_1 \cup T_2) \cap \overline{C}] < \Pr[(T_1 \cap \overline{C_1}) \cup (T_2 \cap \overline{C_2})].$$

This inequality can be interpreted as follows: in the case of an alternative of two threats, it is better to handle both threats together with one single countermeasure than to mitigate the risk of each threat with a separate countermeasure.

Implementation Aspects

This section provides some details on the lessons learned from the implementation of the simulation system.

Rapid Prototyping with Python

The first step in the implementation process was a rapid prototyping approach with Python[2]. The application was designed in an object oriented fashion. As expected, the development was done in short period of time.

While working with Python, we benefited from the simplicity of this programming language. Especially the dynamic typing of the variables during run-time and the required formatting of the code with indentations supported to make a good progress. Another advantage are the built-in data structures such as lists and dictionaries which simplify the implementation of common algorithms such as graph algorithms.

The Python ecosystem consists of lots of additional packages which can be installed easily with the Package Installer for Python (PIP). Two of these packages

[2]Webpage: https://www.python.org

turned out to be very useful in our software protect and are described briefly in the following.

In the development of the simulation system the possibility of cloning an attack defense graph was needed. This is, an exact copy of the graph had to be created which is in a different memory location than the original one. Thy Python module *copy*[3] provides with the command deepcopy() exactly the required functionality. The command recursively creates a copy of an object and all the objects it does contain.

In order to simulate an attack defense graph with different node assessments, it is necessary to iterate through a given range of capability or difficulty values. For example, in the above use case, the impact of an anti-malware toolkit (node N0004) was analyzed by simulating the attack defense graph for each of the capability difficulty pairs from in the set $\{3,4,5,6\} \times \{2,3,4,5,6\}$. Programming in Python, this can be done easily by using the module *itertools*[4] which provides building blocks for iterators based on a given set of values. This module helped a lot in creating different simulation scenarios.

After the prototype of the simulation system was completed, it was used to analyze several use cases. While working with the software, several drawbacks of the Python language did arise. At first, since Python is an interpreted programming language, many errors in the code pop up during the execution of the program. Furthermore, the execution of the interpreted code is slow compared to the code of a compiled programming language such as C++ or Java.

Another disadvantage is the Python global interpreter lock (GIL), which controls the execution of threads in such a way that only one thread is executed at a point of time. As a consequence, implementing a multi-threaded simulation approach does not improve the performance of the simulation system, since at a time only one thread can be executed. The power of a multi core CPU is not maxed out because only one core is used. A solution is the implementation of a multi process approach with interprocess communication which is more complex compared to multithreading. For more details, we refer to (Gorelick and Ozsvald, 2014).

Besides of its drawbacks, rapid prototyping with Python was the right decision, because the simulation tool could be put into work within a very short period of time. It provided a lot of knowledge which turned out be useful in further progress of this project.

Re-Implementation with Java

The major drawback of the Python implementation was its mediocre performance and the restricted support of multithreading. Hence, we decided to do a re-implementation with the programming language Java[5]. Java was chosen because of its platform independence and the large availability of third party

[3]Webpage: https://docs.python.org/3/library/copy.html.
[4]Webpage: https://docs.python.org/3/library/itertools.html.
[5]Webpage: https://www.oracle.com/technetwork/java/index.html.

software packages. The re-implementation started with Java 8. Later, we switched to Java 11, the current version with long term support.

The project was designed as a multi-module architecture which made use of the Java Platform Module System (JPMS) which was introducted in Java 9. Simply spoken, JPMS enables the separation of the code into several modules. Each module must have an unique name and must specify the dependencies on other modules. In particular, it must be specified which elements of the module are accessible. The benefit of this approach is an increase of reliability of the software and a better encapsulation of the software packages. The interested reader finds more information on Java modules in chapter 12 of (Flanagan and Evans 2018).

Since the Java SE Development Kit does not include a build tool which automatically takes care of module dependencies, Maven[6] was chosen as the software management and build toolkit. The layout of the project was a multi-module one. For each module, a project object model (POM) had to be created. The POM includes, among other things, the dependencies of the module and the instructions to build and package the module. The format of a POM file is XML.

The re-implementation with Java resulted in a software with improved running times and proper multithreading support. Compared to Python, the development in Java was more time-consuming. Since Java did not provide modules with the functionality of Python’s copy and itertools modules, additional work had to be spent on implementing these features.

Conclusion

This paper describes the application of a risk assessment model to the use case of a threat induced by ransomware attack. The model is based on attack defense graphs and Monte Carlo simulations. The model was successfully implemented with Java. The analysis of the ransomware use case demonstrated how to apply the model to practical problems arising in the area of cyber security risk assessment. The use case shows how the model can help security specialists to find out appropriate countermeasures to mitigate common threats on computer systems. The effort of applying this model is moderate compared to other risk assessment methods. As a consequence, the model is appealing to small and medium-sized enterprises which can use the model for decision-making on it security solutions with moderate costs.

[6]Webpage: https://maven.apache.org.

References

Blakley B, McDermott E, Geer D (2002) Information Security is Information Risk Management. *Proceedings of the 2001 Workshop on New Security Paradigms*.

Cremonini M, Martini P (2005) Evaluating Information Security Investments from Attackers Perspective: the Return-On-Attack (ROA). In *4th Workshop on the Economics on Information Security.*

Edge KS, Dalton GC, Raines RA, Mills RF (2006) Using Attack and Protection Trees to Analyze Threats and Defenses to Homeland Security. *Military Communications Conference, Milcom 2006*, IEEE.

Ericson CA (1999) Fault Tree Analysis - a History. In *17th International System Safety Conference*.

Flanagan D, Evans B (2018) *Java in a Nutshell* (7th ed.). O'Reilly Media, Inc.

Forum of Incident Response and Security Teams (Ed.). (2019). *Common Vulnerability Scoring System v3.1: Specification Document*. Retrieved from https://bit.ly/2m8c Ssz.

Fraile M, Ford M, Gadyatskaya O, Kumar R, Stoelinga M, Trujillo-Rasua R (2016) Using Attack-Defense Trees to Analyze Threats and Countermeasures in an atm: A Case Study. In *The Practice of Enterprise Modeling, Poem 2016:* 326–334. Springer.

Gorelick M, Ozsvald I (2014) *High Performance Python*. California: O'Reilly.

Hänisch T, Karg C (2019) *Using Monte Carlo Simulation to Estimate the success of it Security Measures in Industry 4.0 Environments*. Presented at 15th Annual International Conference on Information Technology & Computer Science, 20-23 May 2019, Athens, Greece.

Kordy B, Mauw S, Radomirović S, Schweitzer P (2014) Attack–Defense Trees. *Journal of Logic and Computation* 24(1): 55–87.

Kumar R, Stoelinga M (2017) Quantitative Security and Safety Analysis with Attack-Fault Trees. *18th International Symposium on High Assurance Systems Engineering,* IEEE.

Lund MS, Solhaug B, Stølen K (2011). *Model-Driven Risk Analysis: The CORAS Approach*. Berlin, Heidelberg: Springer.

Mauw S, Oostdijk M (2006) Foundations of Attack Trees. In: Won D.H., Kim S. (eds) *Information Security and Cryptology - ICISC 2005*. ICISC 2005. Lecture Notes in Computer Science, vol 3935. Berlin, Heidelberg: Springer.

OWASP (Ed.). (2019, June 27). *OWASP Risk Rating Methodology*. Retrieved from https://bit.ly/1BJAUe8 [Accessed 13 July 2019].

Schneier B (1999a). Attack Trees. *Dr. Dobb's Journal of Software Tools* 24(12): 21–29.

Schneier B (1999b) *Attack Trees*. Retrieved from https://bit.ly/2IcpbcC

Vesely W E, Goldberg FF, Roberts NH, Haasl DF (1981). *Fault tree handbook* (No. NUREG-0492). U.S. Nuclear Regulatory Commission, Office of Nuclear Regulatory Research, Division of Systems; Reliability Research.