PhD-FSTC-2018-45
The Faculty of Sciences, Technology and Communication

Dépt. Systèmes Réseaux, Cybersécurité et Droit du numérique

DISSERTATION

Defence held on 26/06/2018 in Luxembourg

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

AND

DOCTEUR DE L'ECOLE NATIONALE SUPÉRIEURE MINES-TÉLÉCOM ATLANTIQUE BRETAGNE-PAYS DE LA LOIRE - IMT ATLANTIQUE
sous le sceau de l'Université Bretagne Loire

EN INFORMATIQUE

by

Steve MULLER
Born on 24 April 1991 in Luxembourg (Luxembourg)

RISK MONITORING AND INTRUSION DETECTION FOR INDUSTRIAL CONTROL SYSTEMS

Dissertation defence committee

THESE DE DOCTORAT DE

L'IMT ATLANTIQUE (1)
COMUE UNIVERSITE BRETAGNE LOIRE

ECOLE DOCTORALE N° 601 (3)
Mathématiques et Sciences et Technologies de l'Information et de la Communication (4)
Spécialité : Informatique (5)

Par
Steve MULLER

Risk Monitoring with Intrusion Detection for Industrial Control Systems (7)

Thèse présentée et soutenue à Luxembourg, le 26 juin 2018 (8)
Unité de recherche : OCIF (9)
Thèse N° : 2018IMTA0082 (10)

Rapporteurs avant soutenance : (11)

Valérie Viet Triem Tong	Associate Professor, CentraleSupélec
Jean-Marie Flaus	Professeur, Université de Grenoble Alpes

Composition du Jury :

Jacques Klein *Président*	Senior Research Scientist, Université du Luxembourg
Romaric Ludinard	Associate professor, IMT Atlantique
Valérie Viet Triem Tong	Associate Professor, CentraleSupélec
Jean-Marie Bonnin *Directeur de thèse*	Professeur, IMT Atlantique
Yves Le Traon *Co-directeur de thèse*	Professeur, Université du Luxembourg

Invité

Carlo Harpes	Managing director, itrust consulting

Titre : Surveillance des risques avec détection d'intrusion pour les systèmes de contrôle industriels

Mots clés : gestion de risque en temps réel, surveillance des risques, modélisation de dépendances, systèmes industriels, détection d'intrusions

Résumé : Les cyberattaques contre les infrastructures critiques telles que la distribution d'électricité, de gaz et d'eau ou les centrales électriques sont de plus en plus considérées comme une menace pertinente et réaliste pour la société européenne. Alors que des solutions éprouvées comme les applications anti-malware, les systèmes de détection d'intrusion (IDS) et même les systèmes de prévention d'intrusion ou d'auto-cicatrisation ont été conçus pour des systèmes informatiques classiques, ces techniques n'ont été que partiellement adaptées au monde des systèmes de contrôle industriel. En conséquence, les organisations et les pays font recours à la gestion des risques pour comprendre les risques auxquels ils sont confrontés. La tendance actuelle est de combiner la gestion des risques avec la surveillance en temps réel pour permettre des réactions rapides en cas d'attaques. Cette thèse vise à fournir des techniques qui aident les responsables de la sécurité à passer d'une analyse de risque statique à une plateforme de surveillance des risques dynamique et en temps réel.

La surveillance des risques comprend trois étapes, chacune étant traitée en détail dans cette thèse: la collecte d'informations sur les risques, la notification des événements de sécurité et, enfin, l'inclusion de ces informations en temps réel dans une analyse des risques. La première étape consiste à concevoir des agents qui détectent les incidents dans le système. Dans cette thèse, un système de détection d'intrusion est développé à cette fin, qui se concentre sur une menace persistante avancée (APT) qui cible particulièrement les infrastructures critiques. La deuxième étape consiste à traduire les informations techniques en notions de risque plus abstraites, qui peuvent ensuite être utilisées dans le cadre d'une analyse des risques. Dans la dernière étape, les informations collectées auprès des différentes sources sont corrélées de manière à obtenir le risque auquel l'ensemble du système est confronté. Les environnements industriels étant caractérisés par de nombreuses interdépendances, un modèle de dépendance est élaboré qui prend en compte les dépendances lors de l'estimation du risque.

Title : Risk Monitoring with Intrusion Detection for Industrial Control Systems

Keywords : real-time risk management, risk monitoring, dependency modelling, industrial control systems, intrusion detection

Abstract : Cyber-attacks on critical infrastructure such as electricity, gas, and water distribution, or power plants, are more and more considered to be a relevant and realistic threat to the European society. Whereas mature solutions like anti-malware applications, intrusion detection systems (IDS) and even intrusion prevention or self-healing systems have been designed for classic computer systems, these techniques have only been partially adapted to the world of Industrial Control Systems (ICS). As a consequence, organisations and nations fall back upon risk management to understand the risks that they are facing. Today's trend is to combine risk management with real-time monitoring to enable prompt reactions in case of attacks. This thesis aims at providing techniques that assist security managers in migrating from a static risk analysis to a real-time and dynamic risk monitoring platform.

Risk monitoring encompasses three steps, each being addressed in detail in this thesis: the collection of risk-related information, the reporting of security events, and finally the inclusion of this real-time information into a risk analysis. The first step consists in designing agents that detect incidents in the system. In this thesis, an intrusion detection system is developed to this end, which focuses on an advanced persistent threat (APT) that particularly targets critical infrastructures. The second step copes with the translation of the obtained technical information in more abstract notions of risk, which can then be used in the context of a risk analysis. In the final step, the information collected from the various sources is correlated so as to obtain the risk faced by the entire system. Since industrial environments are characterised by many interdependencies, a dependency model is elaborated which takes dependencies into account when the risk is estimated.

Résumé long

La technologie de l'information est devenue une partie importante de nombreux aspects de notre vie – nous comptons sur elle pour coordonner le transport des marchandises et des personnes, pour contrôler les infrastructures, pour gérer les entreprises, pour communiquer et pour nous divertir. Même si la technologie de l'information est un élément essentiel de la plupart des systèmes et des plates-formes, la sécurité n'a pas nécessairement été prise en compte lors de leur conception, en particulier dans l'industrie. Ce manque de sécurité n'a pas été un problème jusqu'à il y a quelques décennies, lorsque les technologies de l'information – ainsi que leur (probablement) plus grand succès, Internet – ont été mises à la disposition du grand public, et tout le monde pouvait accéder chaque système dans le monde entier.

Beaucoup d'efforts ont été déployés pour sécuriser les systèmes vulnérables depuis. Mais même aujourd'hui, beaucoup d'entre eux sont loin d'être protégés de manière raisonnable contre divers types de menaces. En conséquence, les utilisateurs et les opérateurs doivent être conscients des considérations de sécurité résultant de l'utilisation ou de la dépendance de tels systèmes, et envisager des contrôles de sécurité appropriés pour les risques les plus graves.

Pour une organisation, la sécurité doit être abordée à deux niveaux. D'un point de vue technique, des mesures de protection appropriées doivent être mises en œuvre pour que les services ou les composants du système puissent fonctionner en toute sécurité. Des exemples typiques sont les solutions de surveillance, les systèmes de détection et de prévention, ou les contrôles d'accès. Du point de vue de la gestion, une stratégie de sécurité globale doit être développée pour l'ensemble de l'organisation, qui est chargée de la bonne coordination de tous les processus de sécurité. Même si les deux aspects sont indispensables pour un bon niveau de sécurité, ils interagissent rarement les uns avec les autres : d'une part, la direction ne comprend pas nécessairement les détails techniques, et d'autre part, les personnes sur le terrain n'ont pas nécessairement une vision globale du risque pour l'ensemble de l'organisation.

L'objectif de cette thèse est de fusionner ces mondes complémentaires en fournissant une interface entre les deux. Une telle approche a deux avantages complémentaires. D'une part, les problèmes techniques peuvent être instantanément traduits en notions de risque, ce qui aide la direction à comprendre ce qui se passe sur le terrain. D'autre part, elle présente les implications globales pour l'ensemble de l'organisation et aide ainsi le personnel technique à comprendre les conséquences à grande échelle d'un incident. Dans les deux cas, cette approche permettra des réactions rapides en cas d'attaques ou de fautes.

Pour atteindre cet objectif, cette thèse vise à fournir des techniques qui aident les responsables de la sécurité à passer d'une analyse de risque statique à une plateforme de surveillance des risques dynamique et en temps réel. Elle procède en trois étapes, chacune d'elles décrivant un aspect de la plateforme :

- la collecte d'informations sur les risques provenant des sondes sur le terrain,
- la communication des événements de sécurité de ces sondes à un emplacement central, et
- l'inclusion de cette information en temps réel dans une analyse de risque.

Les interactions entre ces modules sont illustrées dans la Figure 1.

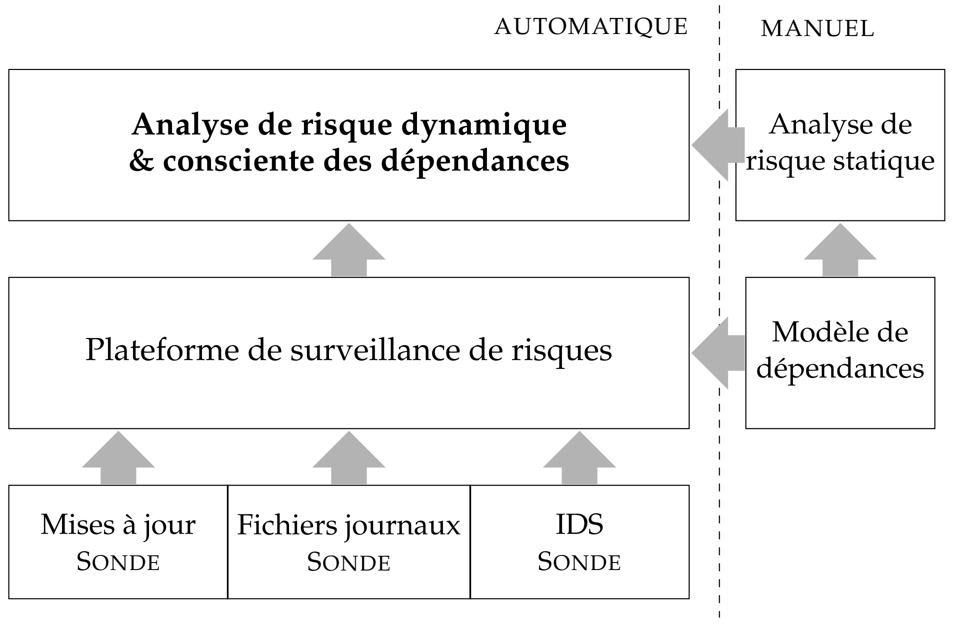

AUTOMATIQUE
MANUEL
Analyse de risque dynamique & consciente des dépendances
Analyse de risque statique
Plateforme de surveillance de risques
Modèle de dépendances
Mises à jour SONDE
Fichiers journaux SONDE
IDS SONDE

FIGURE 1 – Aperçu des composantes du système de surveillance des risques.

En ce qui concerne la collecte d'informations sur les risques, trois exemples concrets de sondes sont développés dans cette thèse. Le premier exemple, un système de détection d'intrusion, détermine le risque que le réseau interne soit compromis. Il applique des techniques d'apprentissage automatique pour détecter les écarts par rapport au comportement "normal" des dispositifs dans un réseau. Le second agent traite les fichiers journaux d'un pare-feu et évalue ainsi le risque des menaces externes. Ces menaces sont liées notamment aux problèmes de disponibilité et couvrent les attaques par déni de service, les réseaux de bot et les renifleurs de port. Le dernier candidat est donné par un outil qui analyse le niveau de vulnérabilité du réseau global. Pour ce faire, il récupère la liste des logiciels installés et les compare aux bases de données de vulnérabilités disponibles publiquement.

La deuxième partie de la thèse porte sur la création d'un lien entre la vision de gestion de haut niveau du risque et les solutions de surveillance techniques. À cette fin, une plate-forme de surveillance des risques est développée, qui sert de référentiel central pour le stockage et la corrélation des risques venant de plusieurs sources. C'est cette plateforme qui joue le rôle de médiateur entre les sondes et l'analyse des risques. Afin d'être compatible avec une grande variété d'agents de risque, un modèle abstrait est introduit qui sert de dénominateur commun pour la communication des

informations sur les risques de façon homogène et cohérente – ce qui est nécessaire si le risque provenant de plusieurs sources doit être corrélé.

Du point de vue de la gestion des risques, il s'avère que les systèmes de contrôle industriels sont caractérisés par la présence de nombreuses interdépendances. Celles-ci proviennent principalement de la complexité technique et de la dimension géographique de l'infrastructure sous-jacente. Cependant, il serait fastidieux de prendre manuellement en considération toutes ces dépendances dans une analyse de risque. Pour cette raison, cette thèse élabore un modèle d'évaluation des risques qui peut être appliqué en plusieurs méthodologies de risque existantes, et qui gère les dépendances. Une approche semi-automatisée est présentée pour générer un tel modèle de dépendance pour une organisation.

Finalement, le modèle de dépendance présenté et la plate-forme de monition des risques sont combinés pour former un système de surveillance des risques en temps réel et sensible aux dépendances. Les concepts développés dans cette thèse sont également appliqués à un outil d'analyse des risques existant, TRICK Service, pour prouver la faisabilité de l'approche.

Abstract

Risk Monitoring and Intrusion Detection for Industrial Control Systems

by Steve MULLER

Cyber-attacks on critical infrastructure such as electricity, gas, and water distribution, or power plants, are more and more considered to be a relevant and realistic threat to the European society. Whereas mature solutions like anti-malware applications, intrusion detection systems (IDS) and even intrusion prevention or self-healing systems have been designed for classic computer systems, these techniques have only been partially adapted to the world of Industrial Control Systems (ICS). This is most notably due to the fact that these industrial systems have been deployed several decades ago, when security was not such a big issue, and have not been replaced since. As a consequence, organisations and nations fall back upon risk management to understand the risks that they are facing.

Today's trend is to combine risk management with real-time monitoring to enable prompt reactions in case of attacks. This thesis aims at providing techniques that assist security managers in migrating from a static risk analysis to a real-time and dynamic risk monitoring platform.

Risk monitoring encompasses three steps, each being addressed in detail in this thesis: the collection of risk-related information, the reporting of security events, and finally the inclusion of this real-time information into a risk analysis. The first step consists in designing agents that detect incidents in the system. They can either interpret the output of existing security appliances (such as firewalls), or monitor (part of) the system on their own. In this thesis, an intrusion detection system is developed to this end, which focuses on an advanced persistent threat (APT) that particularly targets critical infrastructures. The second step copes with the translation of the obtained technical information in more abstract notions of risk, which can then be used in the context of a risk analysis. In the final step, the information collected from the various sources is correlated so as to obtain the risk faced by the entire system.

A novel dependency model ties all parts together and thus constitutes the core of the risk monitoring framework developed in this thesis. The model is loosely based on attack trees, and can be intuitively visualised with boxes and arrows. Despite its visual simplicity, it allows risk assessors to encode the interdependencies of complex risk scenarios, and to quantify the risk originating from the former. While calculations in the model are computationally infeasible, this thesis presents a novel algorithm that provides approximative values for the risk in a very efficient way. The said algorithm opens an entire spectrum of possibilities for computing dynamic risk, which was not possible before.

Acknowledgements

I would first like to thank my supervisors, Prof. Yves Le Traon, Prof. Jean-Marie Bonnin, Dr. Carlo Harpes, Dr. Jean Lancrenon, and Sylvain Gombault for guiding me through the thesis. In particular, I give them great credit for providing valuable and constructive feedback for my thesis, my papers, and my research work in general.

I would also like to thank the members of the jury, Prof. Yves Le Traon, Prof. Jean-Marie Bonnin, Dr. Jacques Klein, Dr. Romaric Ludinard, Dr. Valérie Viet Triem Tong, Prof. Jean-Marie Flaus, and Dr. Carlo Harpes, who have readily declared to concern themselves with the thesis.

I am particularly grateful to Jean and Sankalp for their spiritual support. Our daily discussions helped me stay motivated whenever the thesis did not progress as desired. Furthermore, I am indebted to Matthieu, who gave me valuable tips and research ideas for my work.

Notwithstanding, I owe my deepest gratitude to my sisters, my parents, and my friends. Without them, I would probably not have aimed for a doctoral degree. Thank you!

This work was supported by the Fonds National de la Recherche, Luxembourg (project reference 10239425).

This thesis was co-supervised and financially supported by itrust consulting s.à r.l., who warmly welcomed me at their premises for the entire duration of the thesis and beyond. I'd like to thank them for preparing the topic of this thesis and for enabling me to work in this area.

Their involvement in the national research project *SGL Cockpit* (co-funded by the Ministry of Economy of Luxembourg) and the international research projects *TREsPASS* and *ATENA* (both co-funded by the European Commission) additionally allowed me to work with industrial partners and exchange knowledge with other experts of the domain.

Contents

Chapter 1

Introduction

Information technology has become an important part of many aspects of our life – we rely on it to coordinate transport of goods and people, to control infrastructures, to run businesses, to communicate, and to divert ourselves. Even though information technology is a critical component of most systems and platforms, security has not necessarily been taken into account when they have been designed, especially in industry [1]. This lack of security has not been an issue until a few decades back, when information technologies – as well as their (probably) greatest achievement, the Internet – was made available to the wide public, and suddenly, everyone could potentially access every system world-wide.

A lot of effort has been put into securing the vulnerable systems since. But even today, many of them are far from being sensibly protected against various kinds of threats [1]. As a consequence, users and operators need to be aware of the security considerations resulting from using or relying on such systems, and envision appropriate security controls for the most serious risks.

1.1 Outline of the thesis

For an organisation, security needs to be addressed on two levels. From a technical point of view, appropriate safeguards need to be implemented so that the important business services or system components can operate securely. Typical examples are monitoring solutions, detection and prevention systems, or access control. From a management perspective, a global security strategy needs to be developed for the whole organisation, which is charged with the proper coordination of all security processes. Even though both aspects are indispensable for a good level of security, they rarely interact with each other: on the one hand, management does not necessarily understand the technical details, and on the other hand, people on the field do not necessarily have a global view of risk for the whole organisation.

The objective of this thesis is to merge these complementary worlds by providing an interface between the two. To achieve this, risk assessment methodologies are enhanced in such a way that they include the real-time aspect of security appliances (such as intrusion detection systems, or log files), thus rendering risk assessments *dynamic*. A risk monitoring tool is designed and developed to this end. Moreover, the thesis elaborates several

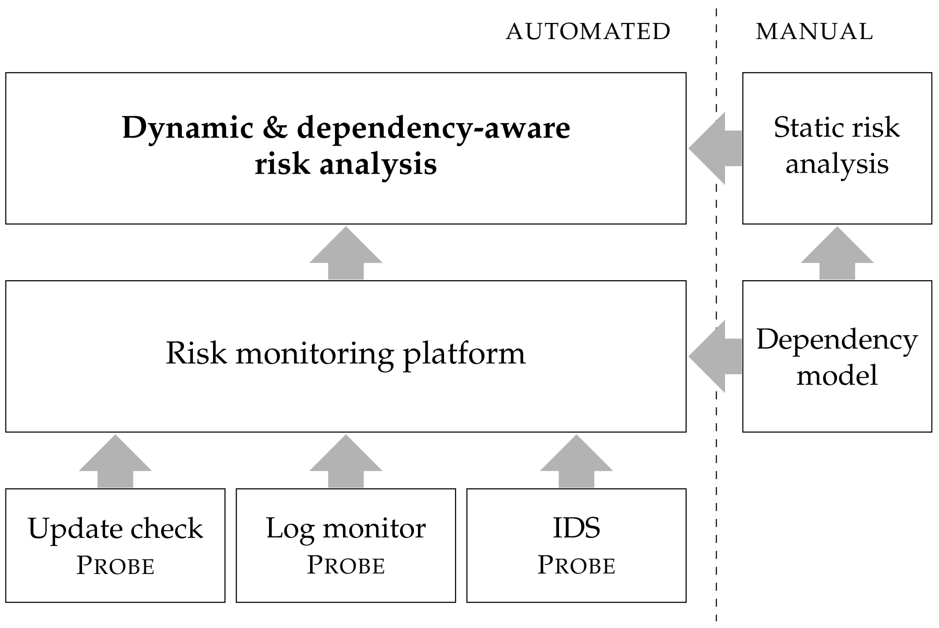


FIGURE 1.1 – Overview of the risk monitoring framework and its components.

risk monitoring agents, including an intrusion detection system, that allow security risk indicators to be reported to the risk monitoring platform.

That way, on the one hand, technical issues are translated into notions of risk, which helps management to understand what is happening on the field. On the other hand, it displays the global implications for the whole organisation, and thus assists technical staff in comprehending the large-scale consequences of an incident.

The components of the framework, and how they communicate with each other, are depicted in Figure 1.1. This piece of work covers three major contributions, each being addressed in a dedicated chapter.

- Chapter 1 (this chapter) introduces the reader to the context and the problem setting, and gives a glance on the achievements made during the thesis.
- Chapter 2 then puts this complete piece of work into the context of other researcher's work, by providing the state-of-the-art of risk monitoring, dependency modelling, and risk reporting (including intrusion detection).
- Chapter 3 presents the risk dependency model developed in this thesis, that helps in associating technical issues to high-level processes.
- Chapter 4 continues the discussion and deliberates how to enhance existing risk methodologies to include real-time aspects. Moreover, this chapter also introduces a risk monitoring platform that translates technical incidents into high-level notions of risk.
- Chapter 5 then deals with the concrete risk monitoring tools that provide the real-time risk information to the latter platform. In particular,

a whole sub-chapter is dedicated to intrusion detection techniques that serve this purpose.

1.2 Context

Perfectly securing a product or a system is practically infeasible, for multiple reasons. On the one hand, to err is human. Past incidents have revealed that security vulnerabilities caused by erroneous implementation or bad configuration are very common. According to [2], 43% of incidents are attributed to malware, 20% to user error, 6% to software bugs); and the complexer a system becomes, the more likely it is that it contains vulnerabilities. This is even more true when components depend on each other. On the other hand, appliances are deployed once and for all after they have been developed, and generally do not evolve – very much in contrast to hackers, who have all the time in the world to find potential attack vectors. Indeed, many security issues are found and fixed only months or years after they have slipped into a product (the average lifespan of a zero-day[1] vulnerability being 130 days [3]).

However, even if it were possible, striving perfect security would not always be sensible. Sometimes the cost of reaching a good security level never pays out, because it is disproportionate to the value that is to be protected. For instance, installing ten different anti-virus solutions on an average home computer would raise the security level (assuming that they all work nicely together), but the added-value of the nine additional ones would be so low that they were not worth the subscription fees. On the contrary, encrypting a hard disk comes at almost no cost, but has considerable security benefits in terms of confidentiality (no unauthorised person can read its contents), so it definitely makes sense to encipher all storage media that contain (or not) sensitive data.

1.2.1 Risk assessments

So, the question when a security measure shall be implemented and when this does not make any sense, is not straight-forward. To begin with, one first needs to know what damage can possibly occur. And this is where risk assessments come into play.

Simply put, a risk assessment is a process that consists in determining all possible risks that could occur, and in estimating their importance. The latter importance is usually expressed in terms of:

- the likelihood of the risk;
- the impact that the risk entails, when it occurs.

Risk assessments can be conducted in either a *quantitative* or in a *qualitative* fashion [4], but mixed strategies exist as well [5].

[1]A zero-day vulnerability is a vulnerability that is discovered before it is publicly disclosed.

	low (1)	medium (2)	high (3)
low (1)	🟩 1	🟩 2	🟩 3
medium (2)	🟩 2	🟩 4	🟥 6
high (3)	🟩 3	🟥 6	🟥 9

FIGURE 1.2 – A very simple heat map based on 3-value impact and 3-value likelihood scales. The risk importance is defined as impact × likelihood. Risks are accepted when their importance is 5 or less (depicted by a green cell); they are to be mitigated otherwise (red cell).

Qualitative assessments describe the risks in words, often with the help of simplified estimation scales for the likelihood and impact (such as 'low', 'medium, 'high'). The objective of such an assessment then primarily resides in defining criteria that state which risks are *accepted*, and which risks shall be *reduced* or *mitigated* (e.g. by implementing appropriate security controls). Although such assessments can be done entirely with the use of words, it is often more convenient to introduce the notion of *risk* or *importance*, which is inferred from the estimated likelihood and impact. For instance, a commonly adopted [6]–[9] (but also criticised [10]) approach is to convert the scale levels to numbers (e.g. 1 to 3), and define the risk as

$$\text{risk} = \text{likelihood} \cdot \text{impact}.$$

In this case, a risk acceptance criterion could be a threshold value for the risk value (e.g. a risk is accepted if its value is higher than 20). That way, one can visualise all the risks in a single matrix, a so-called *heat map*, where one can immediately read off all the relevant information. Various guidance exists on how to create heat maps in general (see e.g. ISO 27005 [11], Appendix A) and for specific use-cases such as privacy impact assessments [12]–[14]. Figure 1.2 shows an example of such a heat map.

Quantitative assessments adopt a more probability-theoretical approach. Their main objective consists in quantifying the likelihood and impact, to make them look like a probability and an expected value. For instance, the likelihood can be expressed as a factor between 0% to 100%, or as the expected number of incidents per time range. The impact usually denotes the financial impact (e.g. in Euro), but depending on the context it might also be encoded as the number of affected people, the market share, or a reputation score. The advantage of such an approach is that the overall risk can be calculated in a very objective fashion. In contrast to qualitative assessments, one does not only speak of *accepting a risk*, but one is also interested in the *residual* risk that remains after security controls have (or have not) been put into place. Since everything is quantified, security controls can be modelled as means to reduce the risk by a given factor (along the lines of "the presence of a firewall reduces the risk of an intrusion by 90%").

Both methods are not mutually exclusive, although one typically prefers one over the other – which method is more suitable, depends on the context, though. In this spirit, a lot of risk assessment methodologies have been proposed, each having its own advantages and fields of application. Prominent examples are CORAS [15], CRAMM [16], EBIOS [17], IT-Grundschutz [18], MAGERIT [19], MEHARI [20], OCTAVE [21], RiskIT [22], but many others exist [23]–[25].

1.2.2 Shortcomings

It's all manual

Even though these methodologies simplify the task by giving advice on how to organise the work, or by providing criteria to come up with good estimates, assessing risk remains a manual process. This is mainly due to the large diversity of organisations that use information systems, and who have entirely different risks and priorities. To prevent risk assessors from missing important risks, international experts have elaborated exhaustive catalogues of risks scenarios. These exist for general information systems [11], [19], [20], but also for more specific domains such as energy providers [26], the Internet of Things (IoT) [27], or data processors [14]. Nevertheless, these catalogues are quite generic and still need to be interpreted in the context of the assessed systems.

Inconsistency leads to errors

There can be many factors which cause a risk scenario, either independently or jointly, so the estimated likelihood has to consider those, as well. The causal chain can in fact be arbitrarily complex, as shows the example of the *Fukushima Daiichi* Nuclear Power Plant disaster in 2011. Due to the earthquake, the power plants were effectively cut off from the grid. Under normal conditions, the reactors are supposed to automatically shut down in that case. However, the accompanying tsunami also damaged the emergency diesel generators, which were positioned too closely to the sea, thus disabling the emergency power supply required to ensure a controlled cool-down of the reactor rods. In consequence, the insufficient cooling lead to nuclear meltdowns, explosions, and release of radioactive material. [28]

Similarly, incidents rarely impact a single part of a system. The former often have additional effects on the environment, on dependent components, or on provided services, which the estimated impact thus has to include.

Despite these facts, risk assessment methodologies don't provide valuable guidance on how risk assessors could encode overlapping risk scenarios, secondary effects, or other forms of dependencies. In contrast, they leave a lot of argumentative freedom and flexibility. As a consequence, the general lack of consistency may lead to double-encoding of scenarios or incomplete assessments, which distort the view on the risk situation of an organisation.

A mere snapshot view of the risk

Since a risk assessment demands a lot of efforts to be conducted properly, frequent updating would neither be feasible nor economically tenable. For this reason, they are generally scheduled on a regular, but large-scale basis (such as once per year). Subsequently, the organisation only obtains a snapshot view of the risk, which may rapidly become obsolete, depending on the context.

Moreover, although risk assessments are useful for determining and mitigating the most critical risks, they are not primarily meant to infer the extent of an incident *when* it occurs. So, if one wishes to gain a real-time view of the current risk, more appropriate models need to be used that also provide up-to-date information.

Insufficient and technical information

As previously noted, estimating the risk is a manual process, and thus subject to personal interpretation. Naturally, the more information is available to the risk assessor, the more accurate his estimation will be.

Many components already provide relevant data that helps to infer the level of risk. For instance, the firewall logs list blocked connections and the overall traffic load, and thus help to rate the exposure to network attacks. In fact, almost any software produces log files, which can be used to detect the overall reliability or threat level of the monitored systems. Furthermore, the (non-) spreading of malware can provide an insight on the effectiveness of cyber-attacks, and the general security awareness level in the organisation. Moreover, the presence of vulnerabilities can be deduced from the version numbers of the installed software and from publicly available exploit information (such as CVE[2]).

There already exist automatic tools that scan the system configuration for security holes and vulnerabilities (e.g. Lynis[3], Nessus[4], or OpenVAS[5]). Nevertheless, even though these tools produce relevant data for risk estimation, actually deducing the risk level is non-trivial. There are three reasons for this.

On the one hand, already processing the data is hard in practise. To start, even if the log files are in an intelligible format, there is sometimes too much data to process, or it contains too much noise so that the important information gets lost. Also, if there is not enough context information, it will be difficult to assess the importance of the risk. This is the case for anti-virus systems, for example. Although the latter are prone to mistakes, they do not provide the confidence that they have in their decision when they raise an alert, which makes it difficult to tell false from true positives. Firewall logs constitute a different example: blocked connections cannot be associated

[2]Common Vulnerabilities and Exposures, https://cve.mitre.org/
[3]https://cisofy.com/lynis/
[4]https://www.tenable.com/products/nessus/
[5]http://www.openvas.org/

to network attacks in a sweeping fashion. That is, an alert does not necessarily signal an intruder, but could also be attributed to a badly configured program.

On the other hand, logging allows administrators to detect *that* a system does not operate correctly, and what circumstances have lead to this state. It does not usually hint at the general exposure to a risk or incidents that may occur in the future. Further efforts have to be made to deduce this kind of information. For instance, the use of statistics or machine learning would allow a system to learn from past happenings and compute the likelihoods of similar, future events.

Finally, risk analyses are meant to give a high-level view of the risk for an organisation. They deal with 'assets', which cover everything that is important to the organisation, including information, business processes, provided services, and infrastructure. The sources of information presented above, however, live in a very technical environment. As such, they are generally bound to a specific hardware or software appliance, and do not 'see' beyond the scope which they operate in. Deducing the risk for the whole infrastructure, with all its interdependencies, is then far from trivial.

1.3 Objectives

This thesis aims at providing a framework for dynamic and dependency-aware risk analyses, which addresses the shortcomings mentioned above. Three major steps are required to achieve this goal.

To begin with, this thesis develops a solid dependency model for encoding related risk scenarios or cascade effects. The strength of the model lies in the fact that it can be applied on top of an existent risk analysis, so that previous assessments do not have to be discarded. Section 1.3.2 below introduces the reader to the problem setting and provides a more detailed description of the sub-objectives for the dependency model.

In a second step, a risk monitoring platform is elaborated, which is able to automatically update a given risk analysis in real-time, thus rendering it dynamic. The core of this framework will be the risk model developed in the first part of the thesis, assuring that dependencies are properly taken into account. The challenges of this task are exposed in the upcoming Section 1.3.3.

Finally, all that remains is the real-time information itself, which is going to be used to infer the real-time risk for the organisation. To this end, not only appropriate tools ('agents') need to be designed that collect relevant information from the field, but also a technique has to be found that allows them to translate the (rather technical) data to notions of risk. Some candidates of such agents are already discussed in Section 1.3.4, together with possibly relevant information that they provide.

1.3.1 Aim for compatibility

The focus of this thesis is clearly put on the development of new techniques for conducting risk analyses, and not on the design of a new risk assessment methodology. Methodologies describe the general work flow of a process and specify the intermediate tasks and their respective outcomes. It is generally up to the actor to decide what work needs to be done, and how. In contrast, techniques are concrete descriptions specifying how such an intermediate task can be realised.

This thesis aims at developing techniques that specify how static risk analyses can be made dynamic and be updated in real-time. For maximum usability, the assumptions made on the underlying methodology are deliberately low. That way, the thoughts developed in this thesis can be applied to a large number of organisations, without requiring them to adapt their processes significantly.

This property may be very important in some contexts, since changing a risk management methodology might not be possible for legal or compliance reasons. Moreover, a complete change of the approach would also implicate that all former material (like existent risk assessments) be discarded.

1.3.2 Dependencies: more structure for risk assessments

An important part of every risk assessment is the proper identification of all relevant assets that are facing risks. The term ‘asset’ is used here to describe any good or service that is valuable to the organisation in question – it can be information, software, material, infrastructure, business processes or services, human resources, or intellectual property. Many, if not all, risk management methodologies feature a description of the asset identification process (see e.g. [15]–[22]). The international standard for guidance on risk analyses, ISO/IEC 31000 [29], also recommends this process to be present. It does not, however, impose to consider how these depend on each other, and what implications these dependencies have in terms of risk. Although this leaves a lot of freedom to the risk assessor, at the same time it does not provide valuable guidance on how to account for dependencies, either.

The following simple example shows that dependencies need to be accounted for, in some way or another.

Example 1.1. *For providers of natural gas there is, on the one hand, the small yet devastating risk of a gas explosion in the transport infrastructure. Such detonations can cause immense material damage, and even casualties. On the other hand, gas providers are bound to strict service layer agreements that require them to deliver the promised amount of gas. In case of failure to do so, they risk to face high fees. These two risk scenarios should definitely feature in any risk assessment for gas providers, no matter the methodology.*

However, one notices that although each of these two can happen independently of the other, there is a chance that an explosion at a critical spot in the gas distribution network would also infer the failure to provide enough gas to a dependent customer, and thus additional financial losses. In the worst case, the cascade effect chain can *even continue. For instance, if the latter customer runs gas turbines to produce electricity, then an outage can be an imminent consequence.*

Careful reflection brings up several possibilities to encode this dependence using 'traditional' means:

- One could completely ignore the fact that one scenario may induce the other, by arguing that cascade effects are much less likely, and thus negligible with respect to the risk scenario itself. This, however, is a questionable assumption; especially for critical infrastructures, cascade effects are not so uncommon [30] and have serious consequences [31]–[33].
- The scope of each risk scenario could be extended in such a way that it would comprise all if its cascade effects. As a consequence, the risk assessment will contain a lot of redundancy, since the effect chains of different scenarios tend to overlap. Moreover, much effort is needed to encode the cascade effects in such a way that they can be consistently updated – if, for instance, the impact estimate changes after security controls have been put into place, one also has to make the appropriate adaptations for each affected risk scenario individually.
- Cascade effects that are shared among multiple risk scenarios could be encoded in a single scenario (e.g. the most suitable one) and be omitted for the others. Even though this approach mitigates some redundancy, it comes at the price of an incomplete risk assessment.

Neither of these methods is particularly satisfying, for they treat all risk scenarios as independent units, and do not account for the case where one incident in the risk analysis gives raise to the other. Moreover, if one wishes to model the dependencies consistently, then extra care has to be taken so that no substantial errors are made during this process. The existence of a method that gives concrete advice will certainly be helpful in those matters.

Modelling dependencies

To add more structure to a possible complex description of the risk situations, dependencies consistent a promising candidate. Building a powerful dependency model is not straight-forward, though. Indeed, assuming that the risk assessment process has identified all relevant assets along with the associated risks, dependencies exist at multiple layers, all of which should be covered.

First of all, individual *assets* can depend on each other to function properly. For instance, a software cannot run if the underlying server machine does not work (see Figure 1.3). To give another example, the billing service for smart grids depends on the correct reading of the smart meters. Similarly, an industrial infrastructure cannot be properly maintained if the required human resources are unavailable, e.g. due to illness or vacation.

Secondly, some risk scenarios may be implicitly contained in other risk scenarios. The loss of data constitutes a good example, which is, indeed,

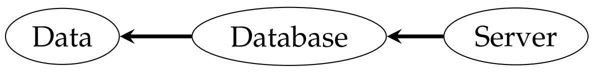


FIGURE 1.3 – Example of functionally dependent assets. An arrow $A \to B$ denotes that problems in A cause issues in B.

a risk scenario on its own – as it can be caused by accidental deletion, hackers, or software errors. However, it also contained in the scenario of a natural disaster in a data centre, which involves the physical destruction of all hardware, and thus also the loss of all data stored on the latter. See Figure 1.4.

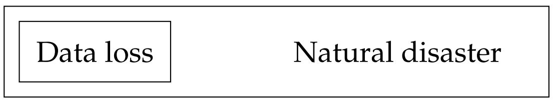


FIGURE 1.4 – Example of implicitly contained risk scenarios. In this case, 'data loss' is implicitly contained in the 'natural disaster' scenario.

Moreover, some security incidents can trigger one or even multiple risk scenarios as a side effect, and may thus lead to a chain reaction. This phenomenon is sometimes referred to as 'interdependencies' [31], and especially constitutes an issue for industrial systems. For example, an electric power producer may depend on a steady supply of natural gas if he uses gas turbines, and technical failures in the gas distribution network may impact the reliability of the electric grid. This is illustrated in Figure 1.5.

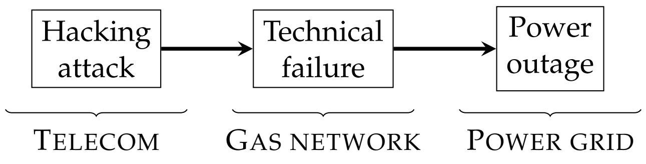


FIGURE 1.5 – Example of interdependencies resulting from security incidents and its cascade effects. An arrow $A \to B$ denotes that problems in A cause issues in B.

A final, yet important case constitute cyclic dependencies. These occur if several appliances rely on each other in terms of security, and a breach at one place has a knock-on effect on other components. In terms of faults, this translates to missing redundancy or bad design; in terms of compromises, it means that there are multiple vulnerabilities that can be exploited to penetrate a system. To give a concrete example, consider a poorly designed web service hosting confidential and valuable data (e.g. medical information) where the administrator can change any user passwords and can retrieve any of the regularly made backups of the user account database. Then unauthorised access to the administrator interface allows an attacker to fetch a back-up, read out and disclose the user passwords, which again leads to unauthorised access. This scenario is depicted in Figure 1.6: note the cycle 'admin interface – backup location – user database' (dotted lines).

In conclusion, the dependency model should be generic enough to cover all of these cases. Chapter 3 will analyse the situation further and work out a

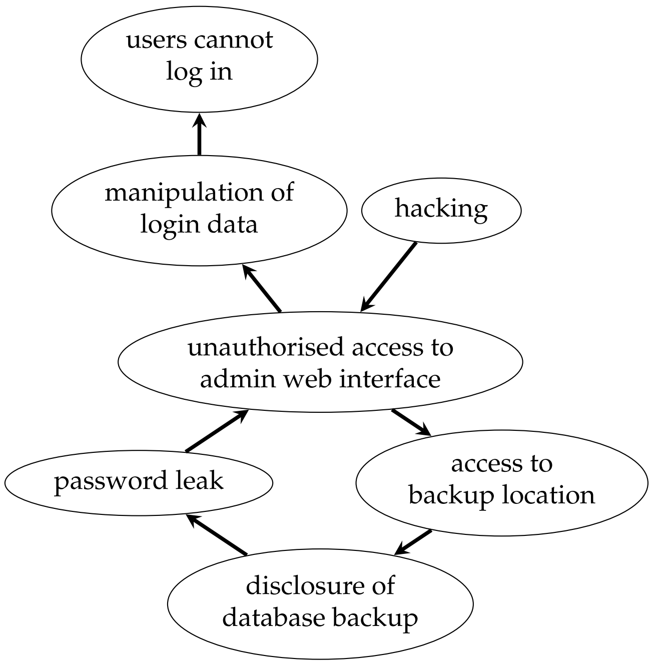

users cannot log in
manipulation of login data
hacking
unauthorised access to admin web interface
password leak
access to backup location
disclosure of database backup

FIGURE 1.6 – Situation where security events are cyclically dependent.

model that does not only fit the needs, but that is also compatible with a wide range of risk methodologies.

Reducing the estimation workload

Risk assessments can be tedious, especially when technical knowledge is required to understand the processes (and the associated risks) of the system, or when the organisation in question is particularly large and complex. In any case, considering dependencies in advance will help to better identify and understand all implications of a risk scenario. This will also greatly improve the consistency of the procedure, since all side effects will be explicitly encoded, helping to avoid redundancy and overlapping scenarios.

In parallel, dependencies already express, to a certain extent, the severity of a risk scenario: intuitively, the more side effects an incident has, the more critical it should be. Also, the more incidents can trigger a risk scenario, the likelier the latter should be. While this is a very rough interpretation, it turns out that in an appropriate model (the one elaborated in Chapter 3), these intuitions can be mathematically founded. The question rises whether this information can also be used to semi-automatically deduce the impact and likelihood of dependent scenarios. If this was possible, a decent amount of work can be saved when assessing the risk, since most of it is already included in the dependency model. In terms of information technology, one speaks of ‘inheritance’ – a set of rules that describe how the impact and likelihood spread from one risk scenario to dependent ones. The precise choice of rules needs to be properly motivated, though, so that it matches reality.

1.3.3 Risk assessment and monitoring combined?

Security has many faces and exists in many contexts, including information technology, the physical realm, politics, and economics. To name a few, information security deals with protecting information from unauthorised use, access, disclosure, manipulation, and destruction. Computer security aims at preventing damage and disruption of computer systems and services. Physical security covers the protection of unauthorised access, theft, or physical damage of personnel, equipment, and entire facilities. National security denotes the security of a nation's citizens, economy, and infrastructure. Finally, economic security copes with solvency, financial stability, and the standard of living.

Especially in information security, security operates on two distinct, but equally important levels. It has a technical aspect, which is in charge of the implementation of appropriate measures on the field; and an organisational part, which makes sure that security is applied where necessary.

On the one hand, technical security usually acts on a specific component of a system (such as a network, a device, an operating system, etc.). It aims at protecting the latter from misuse or damage by accidental or intentional circumstances. Examples include the detection and prevention of threats (firewalls, anti-virus, intrusion detection), the limitation or prevention of vulnerabilities (secure programming, penetration testing), and the striving for resilience (security by design).

On the other hand, organisational security is located more on the management layer, and aims at improving the reliability and resilience of the provided services. By analysing possible threats in advance, and by documenting the procedures to follow, it strives for having all risks for the entire organisation under control at all time. In contrast to its technical equivalent, the focus is put on the fact *that* security is implemented, and less about *how* this is achieved.

Symbiosis

Despite their different nature, both disciplines are complementary; a system can only be secure if both technical and organisational considerations are taken into account. Indeed, just having deployed (technical) protection tools is not sufficient; one also needs to make sure that they are effective. For examples, an anti-virus solution is supposed to detect and neutralise computer infections, but if the subscription is not renewed, or updates are not installed, then the effectiveness of the appliance is very limited. And it is precisely one of the tasks of information security to regularly verify that the implemented security measures are maintained and functional.

In every day business, the effectiveness of a security tool is watched by a responsible system administrator, who is also in charge to fix any issues he encounters. Organisational security, however, is a much slower process, for it requires long-sighted decisions and strategies. While the effectiveness of the defence mechanisms is likely to change every day, with new threats arising, and new vulnerabilities being discovered, risk assessments are comparatively static. In consequence, the risk depicted by such an assessment is thus possibly flawed.

Opportunities

The strong contrast of the up-to-date natures of the two disciplines suggests that linking them is not straight-forward. In fact, it is infeasible for risk assessors to conduct a risk analysis on a short-termed basis, for this is a manual and time-consuming task. However, there is the opportunity to automate at least part of the assessment: since security appliances operate autonomously, and provide threat information in real time, the question rises if the live data cannot be used to automatically update the risk analysis, as well.

Such an approach opens up an entire spectrum of possibilities for risk management.

- First, it eases the work for risk assessors, because they do not need to manually retrieve this information. Furthermore, any previously estimated (and thus subjective) quantities can now be turned into well-founded values.
- The hitherto static risk analysis turns into a risk monitoring tool, that can translate any incident (reported by technical appliances) to a notion of risk. While it helps management staff to understand the consequences of low-level issues, it is also useful to technical staff for identifying the extent of an incident. That way, cascade effects can be prematurely prevented.
- Additionally, a dynamic risk analysis can serve as a simulation tool for determining the effects of hypothetical scenarios. With the help of a dependency model, simulations could moreover identify single point of failures in the infrastructure, or the most critical assets in the organisation.

While solutions for making risk management more dynamic exist[6], some of them are very specific to the context they were developed in. Moreover, to the best of our knowledge, none of the solutions fully address the challenges that are identified below.

Mathematical challenges

The main challenge in linking both worlds will consist in translating reports of incidents to notions of risk exposure. Indeed, security appliances (such as anti-virus engines or intrusion detection systems) report alerts when they believe to have encountered a hostile circumstance. While such an alert certainly hints at an increased exposure to a threat, the absence of any alerts *does not* imply that there is no risk[7]. So the risk exposure cannot be entirely deduced from the fact that an attack has or has not been detected.

[6]The state of the art of dynamic risk management is discussed in Section 2.2.

[7]Risk is about *potential* threats. When there is no attack going on right now, this does not mean that a system is perfectly safe.

In addition, security appliances are programmed or trained to detect undesired states, but they are not entirely fail-safe. Common phenomena such as false negatives (undetected malicious activity) or false positives (alert, but no malicious activity) distort the output of such systems [34]. So, not only does the output not correspond to a concrete risk level, but neither is it particularly reliable. In consequence, unless a protection tool also outputs the certainty of its decision, which can be used in the computation of risk, one has to account for the small error in the risk predictions.

Moreover, some alerts do not signal an on-going attack, but rather hint at suspicious behaviour – which may or may not be related. Examples include an unusually high network traffic rate, failed authentication attempts, disk I/O errors, etc. In contrast to deterministic alerts, such as the detection of a file which is known to contain malware, these increase the overall risk exposure only to a limited extent. When translating alerts to risk, one thus needs to make a distinction between the two.

Technical challenges

Before real-time data can be automatically imported from any probes (such as the security appliances) into a risk analysis, several hurdles need to be overcome.

From a security perspective, any network connection from or to a critical environment constitutes a risk in itself, and shall thus be avoided. However, the probes typically reside in (security-relevant) field networks, and risk assessments are conducted in the less protected office environments. A risk monitoring solution is thus likely to open a back-door for attacks or data leaks, which is what it originally tries to prevent.

But also from a technical point of view, alerts cannot be reported *directly* from the probes to the risk analysis. The reason is a very pragmatic one. Once deployed in the field, the risk monitoring agents are out of control of the risk assessor, and just passively report risk information to a previously defined location. This implies in particular that the latter 'location' is not supposed to change, and thus cannot be the risk assessment itself (which may be versioned, updated, or replaced). It furthermore implies that the reporting format must be clearly defined and cannot be altered at a later stage (much in contrast to a risk analysis, where risk assessors may decide to change the methodology).

For the reasons mentioned above, an intermediate risk monitoring platform is required, that mediates between the two environments. It can, for instance, be installed in the de-militarized zone[8] and operates as follows. On the one hand, it acts as a data storage for the raw data sent by the risk monitoring agents (such as intrusion detection system, firewall, etc.). On the other hand, it computes a level of risk for different scenarios, and serves as repository of real-time information for risk analyses (which will fetch the data they need).

[8]Originally a military term, the de-militarized zone (or DMZ) represents a computer network that separates two environments in such a way that all communication between the two latter pass through the DMZ.

Conclusion

Monitoring tools consistent good candidates for providing relevant information to risk assessments, which thus reflect the real-time risk faced by the monitored system. Several hurdles need to be overcome, both in terms of the mathematical model and in terms of the technical implementation.

Chapter 4 designs a risk monitoring platform that addresses all of these issues. Like the dependency model, it aims to be compatible with as many risk management methodologies as possible. To achieve its goal, it is conceived to operate independently from other risk management processes, and only interferes with the latter by providing appropriate input.

In contrast to approaches by other authors (see state of the art in Section 2.2), the risk monitoring framework presented in this thesis natively supports dependencies.

1.3.4 Obtaining real-time risk information

A risk analysis can be rendered *dynamic* with the help of probes (or *agents*) that continuously monitor the real-time risk situation on the field. The nature of such a probe depends a lot on the kind of risk that shall be retrieved (what risk scenario? what extent – likelihood, impact? what scope?).

Many opportunities exist to turn existent security tools into risk monitoring agents.

- The most simple one consists in observing the log files of any software, and raising alerts when errors (or other suspicious events) are logged. It is to be noted, however, that depending on the quality of the logged content, the obtained indicators may be more or less relevant in terms of risk.
- Sometimes risk levels can also be inferred by actively retrieving data from the environment. Examples include the update status of a software or system, the health of a hardware device, the availability of a service, and more.
- Moreover, it is occasionally necessary to aggregate and correlate risk information from several different components to form an opinion on the overall risk situation.

While it is infeasible to cover all possible situations, a few example cases are presented below, which will be elaborated in greater detail in Chapter 5.

Firewall logs

The purpose of a firewall is to filter all undesired connections and thus to prevent unauthorised people or software from accessing the protected network(s). Firewalls serve as a shield against threats originating from the Internet (in which case one speaks of an 'external' firewall), or as an additional protection inside a network (then referred to as 'internal' firewall).

Indeed, in advanced infrastructures, computer networks are often segregated into smaller parts, each serving a differing purpose (e.g. one for controlling industrial controllers, one for office needs, one for storage, and so on). A central (internal) firewall guarantees that no part can mingle with another one, unless this is explicitly granted. One objective is notably to avoid the spreading of malware from less important to critical parts of the infrastructure.

Since the whole communication traffic flows through the firewall, the latter can deduce very basic statistics so as to see if the bandwidth is saturated, or not. Indeed, many simultaneous connections in a small time window might hint at a so-called distributed denial-of-service (DDoS) attack [35]–[37]. The latter targets at exceeding the capacities of the underlying infrastructure, so as to render it non-operational. Steadily monitoring the number of parallel connections may thus hint at the risk of such an attack.

In contrast to external firewalls, which practically expect to see intrusion attempts, internal firewalls should rarely drop any connections, since the external ones should have prevented any intruders from accessing the network. As a consequence, any connection that has nevertheless been blocked by an internal firewall should immediately raise a red flag and be investigated. Indeed, such an alert could be attributed to the presence of an intruder in the network, or another insider threat (such as a malicious employee).

Patch management

All major operating systems (even Microsoft Windows, to a certain extent[9]) feature a central software repository that lists all programs that have been installed on the device. On UNIX-like systems, packet managers (such as *apt*, *rpm*, *pacman*) make it particularly easy to download, install, and update software on the computer. In particular, verifying if all patches have been applied, is straight-forward.

Moreover, auditing tools exist, such as `linux-exploit-suggester`[10] and `windows-exploit-suggester`[11], that automatically retrieve a list of unpatched vulnerabilities for an operating system.

While system updates are the alpha and omega of a secure environment, it is not always granted that patch management is properly adopted in an organisation. Even worse, practise reveals that in industrial systems, patches are often only applied in irregular or distant time intervals [1]. This can be put down to several causes. Especially in industry, where bugs can have serious or fatal consequences, engineers are reluctant to 'touch a running system', since any change risks to introduce a dysfunction into the system. For the same reason, one often encounters legacy hard- and software in such environments, that is no longer maintained at all [1].

[9]While it does not feature a package manager, it still provides an API for retrieving a list of all installed software.

[10]`https://github.com/InteliSecureLabs/Linux_Exploit_Suggester`. Originally developed by PenturaLabs, the tool has been forked and improved many times since.

[11]`https://github.com/GDSSecurity/Windows-Exploit-Suggester`. Inspired by the related Linux Exploit Suggester, but for Windows operating systems.

The combination of package managers, being able to retrieve the latest updates, and a risk monitoring platform, being able to spotlight the urgency of a patch, thus makes a perfect supporting tool for both patch management and risk management.

Configuration checker

Vulnerabilities do not only originate from security bugs, but also from misconfiguration. A typical example is an unprotected SSH daemon running on a server, or any firewall ports left open. Especially in industrial control systems, where proprietary and custom-developed software are the order of the day, it is virtually impossible to automatically assess the security of the configuration for a complete system. In most cases, a manual audit performed by a domain expert is required. However, some tools exist that can take generic or repetitive tasks: for instance, *Lynis*[12] is an open-source security auditing tool, available for most Linux distributions. It covers standard tests regarding the security of the operating system and some commonly installed applications.

Intrusion detection

Even though security appliances like firewalls, access control and anti-virus theoretically protect a system from external threats, bugs and human failure may lead to the extraordinary condition that the former appliances do not work entirely as expected.

Intrusion detection systems (IDS) are meant to fill the gap between the expected and the real security level. The former continuously monitor a given system (which may be a single machine, a network, or a whole infrastructure) and aim at finding irregularities in what they observe. Even though they are not perfect, either, they constitute a sound tool for administrators to check that other security measures are operating as expected. In addition, when a threat is detected, they sometimes also act on their own and try to neutralise a detected threat – in this case, on speaks of intrusion prevention systems (IPS). Since this thesis does not cover risk mitigation, IPS are not particularly considered in this thesis. It goes without saying that intrusion detection systems are a perfect candidate for risk monitoring agents, since their primary purpose indeed consists in spotting incidents prematurely.

Over the years, many intrusion detection strategies have been developed [38], [39]. Apart from some hybrid approaches (e.g. [40], [41]), intrusion detection systems use one of two modes of operation [42]: *signature-based* solutions proceed by matching their observations with known undesired content, whereas *anomaly-based* systems start by learning the 'normal' (benign) behaviour, and report any deviations from the latter.

On the one hand, signature-based approaches can only recognise misbehaviour that they have been programmed to detect [43]. Their detection

[12] https://cisofy.com/lynis/

abilities is thus quite limited, especially when it comes to sophisticated attacks. On the other hand, anomaly detection systems often use advanced machine-learning techniques, whose decisions might be hard to understand (this is often referred to as the 'semantic gap' [44]) and which can thus be tampered with [45].

As it turns out, when it comes to integrating intrusion detection systems into a risk analysis, several aspects suddenly become important.

- Most importantly, it should be possible to link an alert to a specific risk scenario. Indeed, many machine-learning based systems can recognise anomalous behaviour in a more or less reliable fashion – yet they fail at stating what kind of attack it is, or what consequences it will have.
- Second, the intrusion detection system should not only produce alerts in case of attacks, but also continuously monitor the current situation of risk. The idea is that it should ideally detect an intrusion *before* it actually occurs (when and if this is possible).
- Moreover, since risk management involves decisions by humans, these should also reflect in the intrusion detection techniques. As a matter of fact, machine learning processes take decisions all the time, typically whether to accept a new behaviour or not. To some extent, risk assessors may want to infer with these resolutions, as well.

1.4 Contributions

Parts of this thesis have already been disseminated in national [46] and international [47]–[50] conferences and journals [51], [52].

- The risk dependency model presented in Chapter 3 has been already published in [51] and improved upon in [48].
- Some insights on the risk monitoring platform (Chapter 4) have been presented in [46], [47] and elaborated further in [48].
- Dissemination on an intrusion detection system that is capable of reporting risk information has been made in [52], which covers part of Chapter 5.

The author of this thesis has additionally contributed to:

- the national research project 'SGL Cockpit',
- the Horizon 2020 project 'ATENA' (grant agreement number 700581), partially funded by the European Commission, and
- the European Commission's Seventh Framework Programme project 'TREsPASS' (grant agreement number 318003).

Chapter 2

State of the art

2.1 Dependency modelling

Many authors have identified the need to model dependencies and cascade effects, and they have come up with different approaches. The latter can be classified by the underlying model that is used. In the following, each section introduces one of these and provides the state-of-the-art research that is relevant in the context of this thesis.

2.1.1 Asset diagrams

Simply put, an asset diagram visually represents how assets are linked to each other. In the simplest case, such a diagram is just a hierarchy that serves a purely informational purpose.

In more complex cases, such diagrams can be used to infer risk. For instance, Xiaofang et al. [53] classify assets into three layers, namely business, information and system. On the lowest layer, risk is computed traditionally as risk $=$ impact $\times$ likelihood. Dependencies appear in the model as weighted impact added to the risk of dependent higher-level assets. However, the model is not flexible enough when it comes to modelling the interdependencies of system components in an industry environment. Moreover, the risk formula lacks any real-world interpretation.

The CORAS methodology [15] uses asset and threat diagrams to visually depict and analyse the risk situation [54]. While the purely qualitative approach can be incredibly helpful for identifying and understanding the risk scenarios, some quantification is needed for risk monitoring. Also, the CORAS model is *too* flexible in the sense that an automated system would be unable to process the produced diagrams[1].

Breier [55] improves on the idea to encode information security assets and their dependencies in a tree structure. His model makes use of logical AND and OR gates to make assets inherit the risk from their parent assets.

Other authors have also attempted to deduce risk notions from an asset diagram. Among them, Liu et al. [56] impose additional structure on the asset model and provide overly complex formulas to deduce the risk. Suh

[1]For instance, conditional dependence is encoded as free text in the diagram, which a computer program would need to 'understand'.

et al. [57] do not impose any structure on the asset diagram, and define the risk as the maximum risk of all dependent assets. While this may be an easy approach, it fails when complex (such as cyclic) interdependencies are considered.

In conclusion, asset diagrams excel *per se* by their simplicity. However, in practise, they are not flexible enough to support the encoding of multiple risks per asset, in which case a complex framework has to be imposed in addition. Therefore, this thesis focuses on elaborating a full-featured model, but describes how the latter can be reduced to a simplified version that is only based on asset dependencies (see Section 3.5).

2.1.2 Attack trees, and related

The objective of an attack tree (also called *fault tree*) is the identification of possible causes for an attack (or a fault in general). The root of such a tree is one attack that one wishes to analyse; its direct children are refinements of this attack in terms of possible causes that could infer it. The refinement process can be iterated arbitrarily often. Child nodes can additionally be combined using logical AND and OR gates.

Much related to fault trees is the analysis of *event trees* [58]. However, in contrast the former, they analyse the possible *consequences* of an event, rather than its causes – the general idea is the same, though. Another variant is the *goal-risk model* [59], [60]: instead of modelling an attack, it describes the business objective and analyses its requirements. Every threat to a requirement is then also a risk for the overall goal.

Schneier [61] was the first to use attack trees in an information security context. While Schneier only described the approach in words, Mauw et al. [62] have formalised it using a mathematical framework. Evans et al. [63] extended the concepts with additional metrics and parameters (such as cost and detectability). Schweitzer [64] and Kordy et al. [65] provide a more general extension (named 'attack-*defence* trees') that allows even better integration with risk frameworks. In fact, they include defence mechanisms that limit the effects of an attack, and thus cover the risk mitigation part. Schweitzer [64] additionally describes how such a tree can be extended with fuzzy logic to quantitatively compute the resulting risk. For large analyses, Baiardi and Sgandurra [66] provide a randomised algorithm that computes the involved probabilities in an efficient way. In fact, their algorithm can even be generalised to arbitrary graphs (see Section 3.3.3 of this thesis).

Several authors have then built upon the idea to introduce further notions of risk. For instance, Ingoldsby [67] shows how attack trees can be combined with quantitative risk analyses, and expresses common risk notions in terms of attack tree vocabulary. A comparable approach is taken by Grunske et al. [68]; they show in addition how a full risk assessment (including risk mitigation) could be conducted using their approach. Similary, Gadyatskaya et al. [49] demonstrate how attack–defence trees integrate with an existing risk analysis that is based on risk-reduction factors. Edge et al. [69] introduce the concept of protection trees, which add the notion of bounded security budget to attack–defence trees and the associated risk analysis.

2.1.3 Attack graphs

However, although attack trees describe causal chains (and thus dependencies, to a certain extent), each attack tree can only cover a *single* risk scenario. For a complete risk analysis, multiple attack trees are thus required. But in order to express interdependencies between these risk scenarios, the model needs to be further generalised to also allow edges between these multiple trees. The generalisation is sometimes referred to as 'attack graphs' (to highlight the link to attack *trees*), which in turn are just special cases of Bayesian networks. The latter are addressed in greater detail in Section 2.1.4 below.

McQueen et al. [70] introduce 'compromise graphs', which serve a very similar purpose. Instead of working with probabilities (which an attack tree or Bayesian network would require), they analyse the time that it takes to compromise a target (i.e., the time for an attack to succeed) for highly dependent components in an industrial environment.

Similarly, the risk assessment methodology supported by the Spanish government, *MAGERIT*, also deals with asset dependencies, which are embedded into a graph (see [19], Section 8 "Practical Advice"). But instead of linking assets, it links their security objectives (such as confidentiality, integrity and availability), whenever they have an impact on each other. While such an approach is much closer to a real dependency-aware risk analysis, no explicit formulas are provided for quantifying the risk. This thesis will build up on the idea, though.

A similar strategy is adopted in the case study by Utne et al. [71], who focus on cascading effects in critical infrastructures. They proceed by analysing the interdependencies between several high-level services (such as electricity) and their impact to the society. Their model is called 'cascade graph', but it serves the same purpose than an attack graph. They also provide a framework to quantise the several magnitudes involved in the risk assessment, allowing an explicit computation of risk. Most interestingly, the risk itself is expressed as expected number of people affected by an incident, showing that a quantitative analysis does not necessarily have to be expressed in financial terms.

Kawn et al. [72] combine all of these ideas and provide a generic framework for describing the interdependencies of project management risks. While not directly related to information security, the same principles can be applied. They introduce 'risk dependency graphs' (based on attack graphs) for computing the resulting risk with respect to multiple metrics.

2.1.4 Bayesian networks

Bayesian belief networks originate from probability theory, and primarily serve to compute (conditional) probabilities for dependent events. They use directed acyclic graphs (referred to as the 'causal graph') to represent the events (the nodes) and their dependencies (the edges). Since attack trees are a special case of a causal graph, a good part of the former theory can be generalised to Bayesian networks.

Bayesian networks are an improvements over attack graphs, in the sense that there is a mathematically sound framework for calculating probabilities. Dantu et al. [73] show how Bayesian networks can be used to compute the success probability of an attack in an attack graph.

A risk analysis is not only about determining probabilities, though. Fenz et al. [74] additionally describe a way of deducing the related risk from such a Bayesian network. Rahmad et al. [75] then apply the latter findings also on existing risk methodologies such as MAGERIT [19]. However, they use an exhaustive list of threat scenarios instead of security objectives, which considerably increases the size of the model. This thesis further generalises this concept to arbitrary and user-definable security incidents.

Independently and in parallel, Poolsappasit et al. [76] provide a slightly different approach of modelling risks in a Bayesian network. While also relying on conditional probabilities, they additionally analyse the situation when a particular event is *known* to happen (i.e. its probability being 1) – because it has been observed, for example. This idea is also pursued in this thesis.

2.1.5 Cyclic dependencies

As argued in Section 1.3.2, the dependency model should also support the cyclic nature of interdependencies, which occur especially (but not exclusively) in the industrial domain. However, in contrast to attack graphs, Bayesian networks are restricted to *acyclic* graphs – a limit imposed by the underlying mathematical model.

Bayesian networks can thus not be used as-is. Therefore, Homer et al. [77] adapt the concepts and algorithms to the realm of risk assessments and apply a graph unfolding technique to generalise the model for *cyclic* dependency graphs. Unfortunately, the running time of this process is exponentially large in general, and thus only works for small or sparse graphs.

Another (but related) solution to this problem has been described by Kotzanikolaou et al. [78]. Instead of considering all possible dependencies at once, they split them up into 'n-order dependencies', each occurring after the other. That way, cycles are removed and normal Bayesian theory can be applied. However, the approach involves a lot of additional manual work, since a human has to furthermore specify the order of each dependency.

As a workaround to the entire problem, Wang et al. [79] provide a simplified (and efficiently computable) probability metric for Bayesian networks, which can be generalised to cyclic graphs, as well. However, this comes at the cost of poorly justified choices (in fact, their metric does not properly take dependent events into account), resulting in probabilities that do not necessarily reflect reality.

Since workarounds seem to involve the loss of the mathematical soundness, this thesis adopts a similar approach to Homer et al. [77], but improves on it by generalising the approach and by providing efficient algorithms for computing the risk. The details can be found in Section 3.3.

2.2 Dynamic risk analysis

Typically, a risk analysis is conducted once, and manually updated afterwards if necessary. Dynamic risk management addresses the question how the update process can be automated.

2.2.1 Theory

Based on Hidden Markov Models. Several authors have considered this issue already, and came up with different solutions. Among them, Årnes et al. [80] introduce a risk assessment model based on Hidden Markov Models (HMM). Each asset is represented by a finite state machine consisting of three states – 'good', 'under attack', and 'compromised' –, each involving a certain risk impact. Hidden Markov Models are then used to infer the (probabilistic) current state from observations (such as alerts from an intrusion detection system), and thus to their risk level. While dependencies are not supported, a later contribution by Haslum and Årnes [81] adds at least support for multiple sensors.

Tan et al. [82] improve on Årnes' findings by generalising the approach to arbitrarily many states and continuous-time Hidden Markov Models; moreover, they show how the method can be integrated into a complete risk methodology.

In a similar spirit, Kanoun et al. [83] start from attack graphs, and model each relevant sub-graph as a Hidden Markov Model. Observations from several agents (including bot detections, spam engine, SIP entity discover) are fed to the HMM to infer the probabilities for the specific sub-graph, and thus also for the entire attack graph. Moreover, they describe how the whole framework can be used for running attack simulations, and how it serves as decision support for risk mitigation.

Based on attack graphs. Jahnke et al. [84] start from a dependency graph that links all dependent components of a Voice-over-IP service (including hard- and software). They express risk in terms of availability, integrity, and confidentiality scores for each component – the respective quantity is either measured by a monitoring utility (such as an intrusion detection system), or calculated as a weighted average of its children if no such utility is available.

Instead of coming up with a new approach, Noel et al. [85] generalise the theory known from attack trees to arbitrary attack *graphs*. Moreover, they provide a (randomised) algorithm which computes the probabilities of dependent events – which is no longer straight-forward in the case of arbitrary graphs. The authors additionally define the notion of 'return on investment' that serves as decision support for risk management.

Poolsappasit et al. [76] improve on the previous work by fully integrating the assessment of risk mitigation strategies, based on the question which security controls are worth being implemented.

Xie et al. [86] tackle the issue using a different motivation. Indeed, instead of rendering a risk assessment dynamic by augmenting it with real-time alerts, they want to extend an intrusion detection system with a risk assessment. In fact, they generate a Bayesian network from an attack graph that models the detected intrusion, and rely on CVSS[2] metrics to deduce a related risk.

Based on satisfiability. A totally different approach is adopted by Homer et al. [87]. They are less interested in the success probability of an attack, but rather in the effectiveness of security controls. To that end, the latter are first encoded in a special attack–defence graph (which they call a 'proof graph') and then translated to a boolean formula. Each variable state if a specific countermeasure shall be implemented, and each variable comes at a cost. The objective is then to find an minimal-cost assignment of variables that effectively blocks all attacks.

2.2.2 Risk management systems

Most research in the realm of dynamic risk management has consisted in the elaboration of good descriptive models, and some applications in the domain of intrusion detection (e.g. Rheostat [88], a risk-based intrusion detection system for Java applications). Nevertheless, some work has also been made on designing a complete (dynamic) risk monitoring system.

Paté-Cornell et al. [89] describe an entire dynamic risk management platform that takes real-time signals from monitoring agents, feeds them into a risk analysis, and outputs recommendations based on the determined risk. Unfortunately, only a few details are given. In contrast to the outcome of this thesis, the underlying risk model is a simple list of risk scenarios, which is unable to take complex interdependencies into account.

Similarly, Kanoun et al. [90] show how alerts from intrusion detection systems can be incorporated into an existing risk methodology (in their case, MEHARI). Whereas they rely on complex formulae for computation risk-related quantities, their system is able to infer risk automatically after a manual set-up phase.

Dynamic risk management products are often very specific to their realm of application. For example, Giannakis et al. [91] build a real-time platform that monitors the risk associated to supply chain management. When an

[2]Common Vulnerability Scoring System, https://www.first.org/cvss/.

incident is detected somewhere in the supply chain (e.g. manufacturer, logistics), a risk assessment based on return-on-investment is automatically conducted to decide if procurement from additional suppliers is needed, customers shall be informed, or other measures shall be taken.

Similary, Jiang et al. [92] build a system that monitors the real-time risk of pollution of China's third largest river bed by chemical substances. They rely on real-time data from a geographical information system (GIS) and integrate them into a risk framework that is relevant for the health domain.

Although not entirely related to risk management, Dulac [93] designs a framework for analysing the real-time impact of accidents in engineering systems. He bases himself on an accident model (called STAMP) and shows how it can be extended with monitoring agents.

The (on-going) European Commission's Horizon 2020 project 'ATENA'[3] aims at developing a real-time risk monitoring system for industrial control systems in the energy, gas, and water sectors. Their approach is based on a vulnerability management system that receives live risk data from several software agents, including intrusion detection systems, CVE database, and threat indicators on the deep/dark net.

To the best of our knowledge, no risk management system exists that supports both dynamic (real-time) risk *and* dependencies to a satisfactory extent (as described in Chapter 1). This thesis thus aims at combining the state-of-the-art research in dependency modelling and in real-time risk, to yield a dependency-aware risk monitoring platform.

2.3 Intrusion detection systems

Intrusion detection is a quite old research topic (the first papers being published in the 1980's [94] [95]), yet it still constitutes an actively researched domain of computer security, especially in the field of cyber-physical systems such as Supervisory Control and Data Acquisition (SCADA) systems or Advanced Metering Infrastructures (AMI) [96]. Over the past few years, the increasing interest in machine learning techniques led to the development of more sophisticated, so-called *anomaly* detection systems, which learn the 'typical' behaviour of a monitored network or system. That way, they are able to spot deviations from the normal behaviour and thus, to a certain extent, detect previously unseen attacks.

Given the fact that a lot of different IDS strategies have been proposed over the years [38], [42], it is important to choose the one that really suits the needs. For instance, anomaly detection systems typically require the monitored network to be sufficiently static and predictable. While this is not necessarily the case for arbitrary computer networks, cyber-physical systems usually *do* meet this requirement, so a lot of research [96] has been conducted over the past few years in developing and improving on intrusion detection techniques for cyber-physical systems [97].

[3]Grant agreement number 700581. https://www.atena-h2020.eu/

As stated in the introduction (more precisely, in Section 1.3.4), risk-aware intrusion detection systems should not only produce alerts, but also measure the risk level when no attack is being launched. To this end, it is important that the output of the IDS can be linked to specific risk scenarios.

According to Buczak and Guven [98], there exist (at least) the following data mining techniques, suitable for intrusion detection.

- Artificial Neural Networks are excellent candidates for learning and recognising previously learned patterns. However, it is very hard to explain why a certain decision was taken by such a machinery.
- (Fuzzy) Association Rules learn causal correlations of the form 'if A_1 and A_2 and A_3 hold, then B holds as well'. While this approach sounds promising when it comes to explaining why a certain alert was raised, it it not efficient enough to be used in a real-time context [99].
- Bayesian Networks based data mining techniques learn the graph structure from a set of nodes. Unfortunately, the set of nodes strongly depends on the context and need to be composed manually by a security expert.
- Clustering based algorithms aggregate similar patterns into groups. These groups are typically labelled with an intelligible average value, which summarises the group's content. As such, these algorithms are promising candidates for measuring risk, since they summarise the data rather than inferring new knowledge in an obscure fashion.
- Decision Trees combine concepts from Association Rules and Clustering, in the sense that they use nested rule sets to classify behaviour into groups matching these rules.
- Ensemble Learning is a class of algorithms for determining multiple hypotheses that predict the data best. Unfortunately, these algorithms require the training data to be labelled, and thus cannot be used in an unsupervised mode.
- Evolutionary Computation, comprising genetic algorithms, is based on the survival-of-the-fittest principle. As neural networks, their decisions are very hard to explain.
- Hidden Markov Models try to model a system as a finite state machine, and to infer the current state from observations. Like Bayesian networks, they require background knowledge from a security expert, who needs to come up with a good description of a state machine.
- Inductive Learning proceeds in a very similar fashion than Decision Trees, but infers possible *causes* instead of consequences (it is thus a bottom-up approach, in contrast to the top-down approach adopted by decision trees). Just as the former, it requires background knowledge to be defined beforehand. Moreover, whereas it can perfectly be used to analyse an observed incident, it is less suitable for measuring the risk of an intrusion at all times.

- Naïve Bayes applies basic Bayesian theory to learn conditional probabilities of a pre-defined model. Again, this model needs to be elaborated manually.
- Support Vector Machines are a special case of clustering techniques, since they separate data into two groups. As it turns out, normal behaviour cannot only be described by a single cluster. For instance, when it comes to network traffic, one needs to distinguish between several types of streams, including one-time connection, repetitive pinging, data download, and flooding.

Among these, the best candidate for risk-based intrusion detection is data clustering (or related methods), since it is the only technique that is fast enough for real-time detection, that can be trained in an unsupervised manner, and which is based on explainable decisions (i.e. can be translated to notions of risk), at the same time. There are mainly two distinct approaches for cluster analysis: distance-based methods which regroup all data points that are close to each other (with respect to some metric), and density-based methods which forms a cluster once enough data points are in the same spot.

Portnoy et al. [100] were one of the first to describe how intrusion detection can be achieved using cluster analysis in an autonomous fashion. They state that unsupervised learning requires two assumptions; first, the majority of the traffic must be normal, and second, malicious traffic must be statistically different from benign one.

Distance-based. Intrusion detection systems that are built on distance-based cluster analysis often make use of the so-called 'k-means' algorithm or its variants [98]. The latter splits the data set into k clusters so that the inter-cluster distances are the shortest possible. Since the algorithm requires multiple iterations and is thus relatively slow, several authors have combined it with other methods obtain better results in terms of performance [101]–[103].

In contrast, other authors developed a novel intrusion detection systems that is merely based on the principle of k-means.

Among them, Almalawi [104] describes an advanced intrusion detection system for SCADA devices in his Ph.D. thesis,. Before it can be deployed to a SCADA network, the IDS requires a training phase, in which the observations are split into consistent ('benign') and inconsistent ('malicious') behaviour using the distance-based 'k-means' clustering algorithm, which serves as basis for building detection rules.

Elbasiony et al. [105] present a hybrid intrusion detection system that is based on both signature and anomaly detection. For the latter part, they use a variant of k-means that supports giving a different weight to each data point, improving the performance of the algorithm (according to the authors).

Tomlin et al. [106] also improve on the k-means clustering algorithm by feeding the determined clusters into an additional inference system. While the authors do not directly design an intrusion detection system, they apply their enhanced algorithm on logs originating from power system components, and show that their algorithm detects faults and attacks more reliably than comparative methods.

Wang et al. [107] start from a relatively new clustering algorithm, Affinity Propagation, which does not require a fixed number of clusters (as k-means does). The algorithm is applied to self-collected HTTP traffic and the commonly known KDD data set. They conclude that their approach yields better results in terms of performance and efficiency than other clustering methods.

Density-based. In contrast to k-mean and related, there is a whole class of algorithms [98] that consider clusters to be spots in the data which have a high density of points. The first method of its kind was DBSCAN [108]; as opposed to distance-based schemes, it supports arbitrarily shaped, and a variable number of clusters.

Blowers et al. [109] show, for instance, how the DBSCAN algorithm can be used to detect intrusions in a data set such as the KDD. Similarly, Shamshirband et al. [110] apply DBSCAN to wireless sensor networks to detect denial-of-service attacks. To overcome some issues with the quality of DBSCAN, Amini et al. [111] present the HDC algorithm, an improved version of DBSCAN. They show that its detection rate for network intrusions (from the KDD data set) is higher than for competing algorithms.

However, other approaches exist. Leung et al. [112] introduce a novel density-based algorithm, called 'pMAFIA', that divides the space of data points into a self-adapting grid. To overcome performance problems, they additionally rely on a tree structure for organising the data points in each cell of the grid. The authors apply their algorithm on the KDD data set and conclude that it performs better than state-of-the art distance-based algorithms.

Furthermore, Hendry et al. [113] present a very simple density-based algorithm, called 'Simple Logfile Clustering Tool (SLCT)', which they apply to the KDD dataset in order to cluster similar data points. Their criterion for telling benign and malicious behaviour apart is the degree of similarity within a cluster: according to the authors, normal data is much harder to describe than attacks, and is thus more heterogeneous. They therefore introduce a homogeneity threshold based on which a cluster is labelled a normal or suspicious.

Most interestingly, Wang et al. [114] show that one does not need to decide between distance- and density-based. Indeed, they propose a hybrid approach which makes use of aspects from density, k-mean, and k-nearest neighbour. According to them, their approach can be effectively used for intrusion detection; however, the mixture of multiple algorithms also requires determining more parameters, which is not necessarily easy.

2.4 Conclusion

Risk monitoring is not a new concept. A decent amount of research has been invested in the design and development of real-time risk analyses, especially in the realm of industrial control systems (see Section 2.2). However, the developed solutions are often very specific to a use-case, and cannot be easily generalised. Moreover, they often lack support for properly encoding dependencies between arbitrary risk scenarios of assets. In that sense, these platforms are typically meant to investigate a specific risk scenario. In contrast, this thesis aims at creating a risk monitoring framework that serves as a support for the risk management of an entire system.

However, when monitoring the risk of an entire system, the related risk analysis will be large and complex. This applies especially to industrial control systems, which may have a large and widespread network of interconnected devices. Section 2.1 above discusses existing research efforts spent on describing these interdependencies. However, they often fail at finding a good balance between a mathematically sound model, and one which can be described in simple terms. While attack graphs represent a decent candidate that is both simple and sound, they lack the support for mutual dependencies. This thesis thus focuses on generalising attack graphs (which must be acyclic) to arbitrarily shaped graphs, so that even cyclic dependencies can be properly encoded.

The last part of this thesis consists in designing agents that capture information on the current risk in the field. The most prominent example of such an agent are intrusion detection systems; however, the latter have not generally been designed to operate within a risk management system, but to be deployed as a standalone tool, instead. The issue of integrating intrusion detection alerts into a risk analysis is thus further analysed in this thesis.

Chapter 3

Risk dependency model

3.1 Introduction

3.1.1 Motivation

As argued in the introduction, cascade effects constitute a major issue for industrial control systems, for they may have unforeseen and severe consequences. For critical infrastructures this problem is even more emphasised since an entire state relies on its operation. With the launch of 'Smart Grid Luxembourg' – a project which started in 2012 and aims at deploying a country-wide smart grid infrastructure – the Grand-Duchy of Luxembourg is facing new and unexplored risks in the energy distribution domain. Indeed, in contrast to other industrial systems, the electrical grid spreads over a vast area and includes thousands of smart meters and related devices[1], all being interconnected and thus interdependent.

Like any IT infrastructure, smart grids have to face all standard kinds of attacks – including network intrusions, (distributed) denial-of-service, code injection, to name a few. In contrast to the former, however, part of the infrastructure is located in the less protected field (e.g. data concentrators[2]) and even the end-user's household (which is case for every smart meter). In consequence, the smart grid network does not profit from the same physical access restrictions than a traditional data centre.

Since perfect security can never be guaranteed, smart grid operators have to carefully analyse the consequences of a breach into the system, and its impact on the remaining infrastructure components. The choice of the dependency model is thus essential for conducting a proper and complete risk analysis.

3.1.2 Approach

The risk dependency model introduced in this chapter is based on the principle of attack graphs.

[1]In the case of Luxembourg, 300.000 smart meters have been or will be deployed.

[2]Data concentrators are intermediate devices that aggregate the smart meter readings from an entire geographical area, and send the concentrated data to the central system. Their main purpose is the avoidance of a denial-or-service resulting from thousands of smart meters connecting to the central system at the same time.

This choice has two major advantages over other candidates: first, it is generic enough to cover all relevant cases (in contrast to attack trees), and second, it can be represented in a simple, intelligible, and visual form (in contrast to Bayesian networks and Markov models, which require a good understanding of the underlying mathematical principles).

Attack graphs are often used as a purely informational mean to visualise the involved dependencies. In fact, on their own, they are not equipped with any mathematical model for computing the risk involved with the encoded scenarios. Although some authors (refer to Section 2.1.3 for details) have extended them with metrics and formulae, none of the proposals perfectly meets all needs for conducting a risk analysis with support for (cyclic) dependencies *and* real-time monitoring. Section 3.2 below will improve on the work by these authors, and provide a flexible yet practical framework for both modelling *and* computing risk resulting from dependencies.

Some Bayesian theory will be needed to deduce the current risk in real-time, as monitoring agents report indicator values – this is elaborated further in Chapter 4. It is to be noted that risk assessors are not required to have any knowledge of Bayesian theory. The latter will only be used internally by the risk monitoring platform to deduce the involved probabilities.

3.1.3 Terminology

This section defines the risk-related terms that are used throughout the thesis. They are based on the ISO/IEC standard 31000 [29] and are adapted to the concepts introduced in this thesis.

Scenarios and events. Intuitively, a *risk scenario* is the description of a situation that could hypothetically occur in the future (and may have in the past), including all consequences and side effects. In this context, an *incident* refers to the situation where evidence exists that a risk scenario is currently occurring or going to occur. While a risk scenario may be comprehensive and complex, it usually consists of small steps, each either contributing to the consequences of the whole scenario, or being one of its causes. These 'steps' are referred to as *(security) events*. It is up to the risk assessor to choose the level of detail when modelling the individual events. Several events may be combined to a larger one, e.g. to reduce the size of the model – at the cost of precision loss, of course.

In order to provide a better understanding for these notions, a simple example is given below.

Example 3.1. *For a smart grid operator, the availability of the electrical grid is of first priority. It is endangered by many factors, though; since the grid is nowadays controlled by computers, any problems in the data centre (be it of natural or human origin), or configuration mistakes can have devastating impacts. But also physical destruction, such as caused by earthquakes or lightnings, can destabilise the power network.*

In this example, the analysed risk scenario *is the instability of the grid. Hacking attempts, natural disaster, misconfiguration, and data centre in-operability, each*

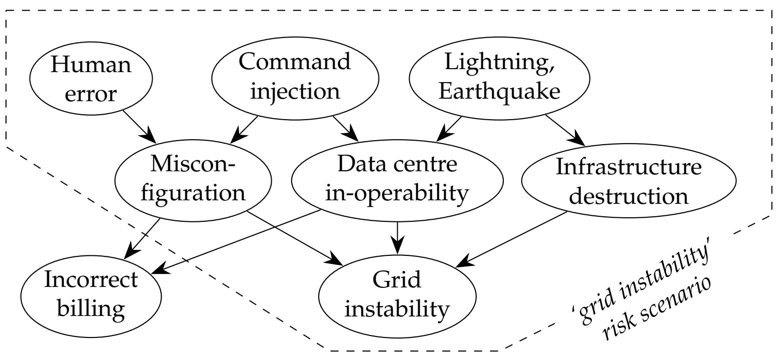


FIGURE 3.1 – Graphical representation of Example 3.1.

constitute an event*. Note that the instability of the grid may also be an event for a different risk scenario, for instance in case a third party is relying on electric current.*

Example 3.1 can easily be represented visually; Figure 3.1 shows how the involved security events can be depicted using a dependency graph. In this representation, risk scenarios are nothing more than sub-graphs, and can thus be visualised by simply drawing a border around the involved nodes. More on this in Section 3.2.

Assets. An asset is any item, physical or virtual, that is of some value to an organisation ([11], §8.2.2). There can be many kinds of assets, including but not limited to the following:

- Data or information can have an important legal, reputational, or operational impact on the organisation when it is corrupted, stolen, lost, or disclosed.
- Business processes may have high risks associated to them, especially if they are part of the core activity of the organisation.
- Assets may also be concrete objects having a monetary value, such as buildings, equipment, hard- and software, or similar.
- Human resources consist another type of assets that can be very valuable to an organisation. This is even more true if the respective job positions own specific domain knowledge that cannot be easily replaced.

Since in most cases, risks are linked to a concrete asset, it is a sensible approach to systematically list all relevant assets and investigate the risks associated to common security aspects (such as confidentiality, integrity, or availability) ([11], §8.2.3).

Security controls. A security control (or *counter measure*) is any mechanism that increases the security of a specific asset or the entire organisation. The purpose of implementing such a control consists in mitigating the occurrence of a risk scenario, or limiting its impact.

3.1.4 Objective

The objective of this chapter consists in designing a dependency model that can readily be used later on, in Chapter 4, when risk is calculated automatically and in real-time. To this end, a decent amount of effort is also dedicated to the elaboration of meaningful formulae and efficient algorithms. Since the model is meant to be used in the context of a risk assessment, this thesis also documents the process of building the dependency graph from existent material.

3.1.5 Outline

The chapter is organised as follows.

- Section 3.2 introduces the dependency model and equips it with notions of risk.
- Section 3.3 provides formulae for determining risk in the context of the dependency model, and presents algorithms for computing it numerically.
- The sensitivity with respect to estimation uncertainties is inspected in Section 3.4.
- Since establishing and maintaining the model might be time demanding for large organisations, a scheme is developed in Section 3.5 for generating it from existent documentation (such as an asset inventory). This procedure can be automatised to a certain extent.
- Subsequently, the hitherto findings are applied to a real-world use-case, namely the smart grid of Luxembourg, in Section 3.6.
- Further extensions and special cases are discussed in Section 3.7.
- The chapter is finally concluded in Section 3.8, highlighting the flexibility of the developed model and its role in a risk monitoring platform.

3.2 Defining the model

A risk analysis consists, among others, in describing and understanding the identified risk scenarios. The underlying risk assessment methodology then dictates how the latter should be identified, evaluated, rated, classified, or sorted. The dependency model developed in this thesis is meant to complement this methodology.

Even though dependencies can help in finding new risk scenarios, this chapter is not about *identifying* them, but rather about analysing how they interact. To this end, the causal chain of each scenario needs to be carefully inspected, so as to extract the relevant intermediate steps. The latter are then explicitly encoded as security events in the dependency model.

It is to be noted that the entire model is based on individual events, and does not have a direct notion of risk scenarios. In particular, whenever dependencies are mentioned, they refer to dependencies *between security events*. Nevertheless, risk scenarios appear in the model as a collection of events and their interactions. In other words, the dependency model provides a more fine-grained description of risk scenarios. In that sense, it is really just a tool extending an existent risk methodology, providing clarification for the encoded information.

3.2.1 Risk

Risk is a three-fold abstract concept that is caused by the simultaneous presence of three circumstances:

- there is an external *threat* (something causing it);
- a *vulnerability* is exploited (something that is at risk);
- the risk has an actual *impact*.

If any of these is missing, the risk is effectively zero. For instance, running vulnerable software is fine if no one can access the computer (i.e. if no threat is present). Similarly, if a software has been patched, it is generally no longer vulnerable to the related exploits and the resulting risk becomes low again. Finally, if vulnerable software is running on an isolated virtual machine, exploits do not have any impact on the real system, so the risk for the latter is comparatively low.

For that reason, risk is commonly defined as a combination of threat, vulnerability, and impact:

$$\text{risk} = \text{threat} \times \text{vulnerability} \times \text{impact},$$

or after replacing each factor by its respective intrinsic property,

$$\text{risk} = \text{likelihood} \times (1 - \text{security}) \times \text{impact}, \tag{3.1}$$

where likelihood $\in \mathbb{R}_+$, security $\in [0, 1]$ and impact $\in \mathbb{R}_+^n$ are further specified below.

Threat. In contrast to other methodologies, the likelihood is *not* defined to be a probability-theoretic number between 0 and 1. In fact, it is meaningless to say that a risk scenario occurs with a certain probability. Indeed, consider the statement "there is a 1% chance of fire", then it is clearly not obvious to determine how often one expects it to happen. Moreover, it is not clear what the upper bound (100%) imposed by probabilities should represent. Instead, a far more intuitive approach consists in making statistical statements, such as "there is a chance of fire for 1 out of 365 days in a year". However, in that case, one really speaks of an expected frequency rather than a probability.

The likelihood is therefore defined to be the *expected number of times* that a risk scenario occurs. This quantity is unbounded and can thus be arbitrarily high. Moreover, it is compatible with Equation 3.1, for an *expected frequency* times the *impact* yields an *expected impact per time unit*.

Likelihoods play a central role in risk monitoring: indeed, any alerts raised by risk agents (intrusion detection systems, firewalls, malware protection, ...) denote the presence of a threat, and thus an increased likelihood of occurrence for the triggered risk scenario. Put differently, it is the likelihood that risk monitoring agents continuously estimate for some specific security events.

Vulnerability. Here, 'security' is a weighting factor between 0 (no security, so maximum exposure to risk) and 1 (perfect security, so no risk). In case one does not wish to bother about vulnerabilities, one can adopt a pessimistic view and set 'security $:= 0$', thus only concentrating on threats.

The 'security' parameter plays an important role for determining the risk mitigation strategy. Indeed, the implementation of security controls induces an increase of the 'security' parameter for the affected assets and related risks. Simulations can then reveal the extent to which a counter measure reduces the overall risk for the entire organisation. This allows risk assessors to decide whether the implementation of that measure is worth the investment and maintenance costs for reducing the risk.

Although risk mitigation is not directly covered by this thesis, Section 3.7.2 gives an insight how risk treatment can be performed with the help of the model. Further details can be found in the literature. For example, Harpes et al. [115] have elaborated a risk assessment method that determines the effect of security controls on the return on security investment (ROSI). To this end, they introduce risk reduction factors, which play a similar role than the 'security' parameter mentioned above.

Impact. Finally, the impact denotes the costs of a risk scenario when it happens. In theory, any unit may be used for specifying the costs. However, most risk methodologies require the impact to be homogeneous throughout the analysis, so as to add them up or compare them. For that reason, the costs often expressed in financial terms (e.g., \$ or €), which is considered as some kind of common denominator. This restriction should not be mandatory, though, because especially in critical infrastructures, safety and availability are far more important than any monetary losses. Several authors [70], [71] have indeed shown that risk analyses can also be conducted in other terms than money.

The model presented in this thesis does not put any restrictions on the impact. In this model, a risk scenario may result in financial loss (unit: \$ or €), casualties (unit: number of people), and legal implications (unquantified) at the same time. Section 3.2.4 formalises the impact in the context of dependency graphs, and show that even inhomogeneous impacts can be used in computations and comparisons.

3.2.2 Dependency graph

All security events are included in a single graph as nodes, and arrows between them are used to represent the causal chains. Such an arrow always denotes a *direct* dependency – indirect effects are not explicitly indicated, even though they can be read off from the graph by following arrows. Some events do not have an antecedent; they are the root causes, that ultimately trigger all other events. In particular, these root causes are considered to be mutually independent.

The general idea behind the dependency graph is centred around the simulation of an incident. Incidents start at a root event, and cascade through the graph, following the arrows. Despite its similarity to Markov chains, a dependency graph does not depict the state of the entire system. Indeed, in the dependency graph, multiple events can occur simultaneously, while a Markov chain is always in a single state. The goal of the simulation is to find out which secondary security events can be reached from that incident, and to what extent.

Most notably, this approach implies that no risk scenario can be more likely than the incident that causes it. Although it sounds trivial and somewhat redundant, this invariant is not guaranteed to hold for all models, especially those that rely on dependency formulae which are based on linear combinations of dependent events.

Formally, dependencies are encoded in a directed graph

$$\mathcal{G} = (V_\mathcal{G}, E_\mathcal{G})\,.$$

The set of vertices $V_\mathcal{G}$ contains all security events that are relevant for the risk analysis. The set of edges $E_\mathcal{G}$ encodes the dependencies themselves; it contains an edge (a, b) whenever a security event $a \in V_\mathcal{G}$ can potentially induce another event $b \in V_\mathcal{G}$. To ease the notation, write

$$a \to b \qquad \Longleftrightarrow \qquad (a, b) \in E_\mathcal{G}.$$

Since the model is built around the simulation of an incident, one is interested in all of its side effects. In graph language, this problem consists in determining all vertices for which there is a path from that certain event. In that spirit, an event a is said to be *eventually causing* event b iff there is a directed path from a to b. Write

$$a \rightsquigarrow b \qquad \Longleftrightarrow \qquad \exists x_0, \ldots, x_n \in V_\mathcal{G} \quad \begin{cases} \forall i < n, & x_i \to x_{i+1} \\ x_0 = a \\ x_n = b. \end{cases}$$

Guidelines

In order to build the graph, one should first identify the risk scenarios that are the most important to the organisation in question. For example, the unavailability of a business-critical service generally features among them.

For smart grids, this translates to the destabilisation of the electrical grid (or power outages). Other risk scenarios may include theft of confidential data, privacy issues, or break-down of expensive equipment (particularly for industrial control systems).

Risk methodologies can support one on determining the risk scenarios. There is an alternative approach that one can adopt. It consists in first listing all assets that are valuable, in some way or another, to the organisation. For each asset in this inventory, one can ask what happens if one of the basic security aspects – confidentiality, integrity, or availability – is violated, and deduce the respective risk scenario. Formally, this amounts to picking a subset

$$V_{\mathcal{G}} \subseteq A \times S,$$

where A is the set of all identified assets, and $S = \{C, I, A\}$ represents the security aspects. To ease the notation, write $a.s$ for $(a, s) \in V_{\mathcal{G}} \subseteq A \times S$. For example, using the syntax defined above, the dependency "hard disk break-down causes loss of data" can be expressed as

$$\text{HARDDISK.A} \to \text{DATA.A}.$$

3.2.3 Degree of dependability

Dependencies may appear to several extents. An event may implicitly induce another, in the sense that if it occurs, the other one is known to occur as well. This is the case for a hard disk, for instance: if it is physically destroyed, then any data stored on it is permanently lost. An event may also cause another one only part of the time (e.g., if the reasons are unspecified or unknown). For example, hot and dry weather may cause forest fire, but not always. Since the exact circumstances may be out of scope of the analysis, or just unknown, the effect can be approximated with a stochastic process, having a certain probability to occur (between 0 and 1).

To encode the degree to which an event *potentially* causes another, define a probability map

$$\begin{aligned} p : E_{\mathcal{G}} &\to [0, 1] \\ (a, b) &\mapsto p(a, b). \end{aligned}$$

For $a, b \in V_{\mathcal{G}}$ and $x \in [0, 1]$, also introduce the notation

$$a \xrightarrow{x} b \quad \Longleftrightarrow \quad a \to b \quad \wedge \quad p(a, b) = x.$$

Using this terminology, Example 3.1 can be visualised in a dependency graph as shown by Figure 3.2. Note that in this description, three incidents (misconfiguration by human error, command injection by a hacker, or natural catastrophes) have been identified to be the root causes of the 'grid instability' risk scenario.

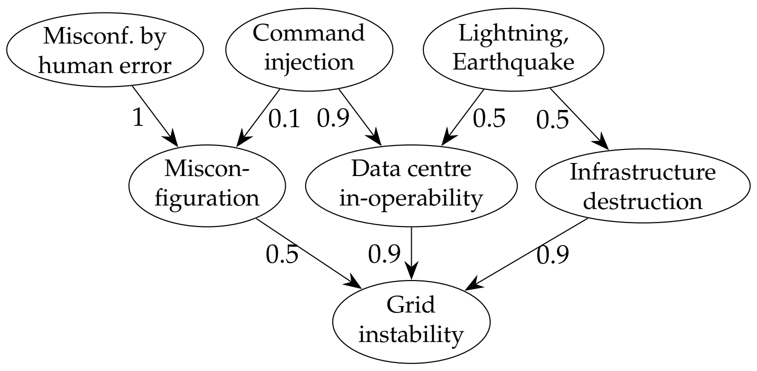


FIGURE 3.2 – Graphical example of a dependency graph with probability values.

$p(a,b)$	Interpretation
0.1	Event a may cause b, but this is judged unlikely.
0.5	Event a may cause b. This is the default.
0.9	Event a causes b in most cases.
1	Event b is implicitly contained in a.

TABLE 3.1 – Possible interpretation of probability values for dependencies.

To ease the work with probabilities, it is recommended to fix a scale in advance, which guarantees that all probability estimations are made in a consistent way. Moreover, the smaller the choice of possible values, the less likely it is to produce inconsistent estimates. An example of such a scale is given in Table 3.1.

It is to be noted that probability values of 1 imply that the dependent event is implicitly covered by (so, contained in) its antecedent. An example would be the break-down of a server, which intrinsically implies its incapacity to operate properly and to provide a certain service. In contrast, even though there is a high chance that a successful intrusion is followed by a takeover of the affected server, it is not absolutely guaranteed that a hacker manages to do so. Moreover, in terms of risk mitigation, it may be possible to set up defensive mechanisms that prevent him from achieving his goal. In such cases, it is recommended to put a probability value less than 1.

There is a reason why this map is called the *probability* map. Indeed, for two events $a, b \in V_{\mathcal{G}}$, a causes b with a certain chance. This random experiment can also be applied to the dependency model, by turning it into a random graph $\mathcal{G}^*$. The latter is like a normal graph, except that its edges are only present with a certain probability – the one specified by $p(\cdot)$. In other words, the new, probabilistic graph $\mathcal{G}^*$ is equipped with a probability distribution $\mathbb{P}$ such that

$$\mathbb{P}[(a,b) \in E_{\mathcal{G}^*}] := p(a,b).$$

In what follows, no distinction will be made between the dependency graph $\mathcal{G}$ and the associated randomised graph $\mathcal{G}^*$, for convenience. The above probability can thus also be written

$$\mathbb{P}[a \to b].$$

Similarly, one can deduce the probability that an event *ultimately* triggers another, that is,

$$\mathbb{P}[a \rightsquigarrow b].$$

However, as it turns out, computing this quantity is highly non-trivial for general graphs. Section 3.3 will discuss this probability further, and will determine efficient algorithms for calculating its value.

Remark 3.2. *Despite the parallels between a dependency graph and a Bayesian network, they are* not *the same. Even though the vertices in a Bayesian network also correspond to events, and its purpose also consists in determining the probability of occurrence for these events, both models operate in a fundamentally different way.*

Bayesian networks are settled around conditional probabilities, which describe the chance $\mathbb{P}[\beta \mid \alpha]$ that a certain event β will occur if another event α has been previously observed. This is different from the definition of the probability map $p(\alpha, \beta)$ introduced above, which indicates the probability that event α causes *β. In the literature [116], the latter is denoted $\mathbb{P}[\beta \mid do(\alpha)]$ or $\mathbb{P}[\beta \mid set(\alpha)]$.*

To see why they are different, consider the following small example. In the chain $\alpha \overset{0.5}{\to} \beta \overset{0.5}{\to} \gamma$, the conditional probability $\mathbb{P}[\gamma \mid \alpha] = 0.25$, since if α occurs, there is a 0.5^2 chance that γ will occur, as well. However, $p(\alpha, \gamma) = 0$, because α does not directly trigger γ.

Bayesian network are used to find out from statistical data which events depend on each other, whereas the dependency graph already encodes this information explicitly.

3.2.4 Risk in a dependency graph

Likelihood. So far, dependency graphs are only centred around causality and the related probabilities, but have nothing to do with risk. For convenience, introduce the map

$$\mathbb{L} : V_{\mathcal{G}} \to [0, +\infty),$$

representing the likelihood of occurrence of each security event. Recall that the latter is expressed as an expected frequency (see Section 3.2.1).

As announced in Section 3.2.2, dependency graphs are meant to simulate an incident that starts at one of the root causes in the model. In that spirit, consider such an incident that starts at $\alpha \in V_{\mathcal{G}}$. In this specific simulation, any other event $\beta \in V_{\mathcal{G}}$ can only be eventually caused by α (if at all). The probability that this happens is exactly $\mathbb{P}[\alpha \rightsquigarrow \beta]$. Since α is expected to occur at a rate of $\mathbb{L}(\alpha)$ by definition, and each time β is triggered with probability $\mathbb{P}[\alpha \rightsquigarrow \beta]$, the expected frequency of β is consequently

$$\mathbb{P}[\alpha \rightsquigarrow \beta] \cdot \mathbb{L}(\alpha).$$

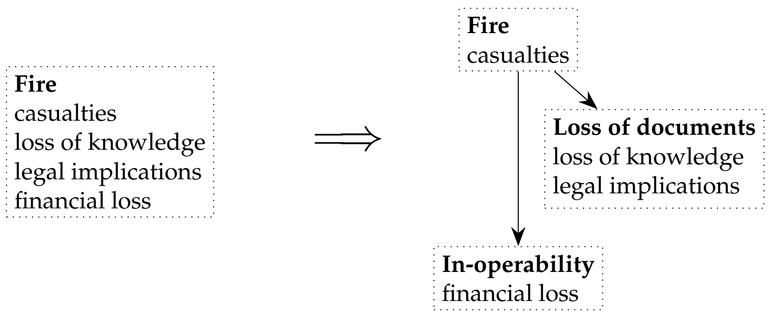


FIGURE 3.3 – Split-up of a risk-scenario in multiple events, with impact attached to the most appropriate one.

Since all root causes are assumed independent, the above finding can be generalised to the complete graph, to yield

$$\mathbb{L}(\beta) = \sum_{\alpha \in \mathrm{rt}(V_{\mathcal{G}})} \mathbb{P}[\alpha \rightsquigarrow \beta] \cdot \mathbb{L}(\alpha),$$

where $\mathrm{rt}(V_{\mathcal{G}}) \subset V_{\mathcal{G}}$ denotes the collection of root vertices (i.e., those without an incoming edge).

Impact. In traditional risk analyses, the impacts of linked risk scenarios tend to overlap, resulting in redundancy and thus in a distorted view of the situation. To overcome this problem, impacts are defined directly for the security event that is immediately responsible for it. Note that this may result in splitting up an impact for a risk scenario, and in distributing it among multiple causes.

Example 3.3. *For example, a fire incident in an office environment has multiple effects: in the worst case it can result in casualties, but it may also lead to temporary in-operability of the service and loss of important documents, with possible legal implications. Since documents can be lost and the service can be stopped for different reasons as well, it may be sensible to extrapolate them in dedicated security events. The overall impact for fire is then split up among the three as shown in Figure 3.3.*

As the example depicted in Figure 3.3 already suggests, the impact should be defined in such a way that the definition does not impose any restrictions on the unit. Indeed, an event may have an impact in terms of casualties, financial, legal matters, or others, all of which cannot be easily compared to one another. However, at the same time, the impact will be used in computations and comparisons afterwards, which requires some compatibility nevertheless.

To address the issue, each type of impact will be described by a set of allowed values, respectively (such as the non-negative numbers $\mathbb{R}_+$). Here, two impacts are considered to be of the same type if they can be compared to some extent (e.g. if they have the same unit, or if one can be converted to the other).

Formally, introduce a collection of sets $I_1, \dots, I_n$, each being linked to an impact type. The impact is then defined to be a map

$$\mathbb{I} : V_{\mathcal{G}} \to I_1 \times \cdots \times I_n$$

that associates an impact for each impact type to every security event in the graph. Note that for a given event, the impact is typically only specified for *one* of the types. There are cases, however, where an event has multiple impacts (such as in the example below).

In Example 3.3 above, there a four different impact types involved.

- $I_{\text{cas}} = \mathbb{R}_+$, the number of casualties;
- $I_{\text{know}} = \mathbb{R}_+$, the amount of knowledge, measured in number of hours required to acquire it;
- $I_{\text{leg}} = \{\text{yes}, \text{no}\}$, whether it has legal implications (the group operation $+$ being the boolean OR in this case);
- $I_{\text{fin}} = \mathbb{R}_+$, financial losses, specified in €.

An impact assessment could look as follows.

$$\begin{aligned} \mathbb{I} : V_{\mathcal{G}} &\to I_{\text{cas}} \times I_{\text{know}} \times I_{\text{leg}} \times I_{\text{fin}} \\ \text{'fire'} &\mapsto (20, 0, \text{no}, 0) \\ \text{'loss of documents'} &\mapsto (0, 160, \text{yes}, 0) \\ \text{'in-operability'} &\mapsto (0, 0, \text{no}, 50.000) \end{aligned}$$

Risk. Recall from Section 3.2.1 that risk is defined as the combination of likelihood and impact. Since impacts constitute general sets (and may only consist of boolean values, `true` or `false`), they cannot be combined as-is with likelihoods. Each impact space I_i thus needs to be equipped with a map r_i that relatives an impact with respect to a likelihood of occurrence. It is required to be of the form

$$*_i : \mathbb{R}_+ \times I_i \to \mathbb{R}_+$$

and is supposed to turn a likelihood (expected frequency) and an impact into an 'expected impact per time unit'. Since for most impact types, the space just equals $I_i = \mathbb{R}_+$, the corresponding map can be defined as

$$*_i : (l, i) \mapsto l \cdot i.$$

For boolean impact types, i.e. $I_i = \{\texttt{true}, \texttt{false}\}$, the map is defined as

$$*_i : (l, i) \mapsto \begin{cases} l & \text{if } i = \texttt{true} \\ 0 & \text{if } i = \texttt{false} \end{cases}$$

These individual maps can be combined to a single map $*$ by component-wise application. More precisely, it is defined as

$$\begin{aligned} * : \mathbb{R}_+ \times (I_1 \times \cdots \times I_n) &\to \mathbb{R}_+^n \\ (l, (i_1, \ldots, i_n)) &\mapsto (l *_1 i_1, \ldots, l *_n i_n). \end{aligned}$$

Intuitively, the map $*$ maps a likelihood and a *combined* impact to a *combined* expected impact per time unit. Put differently, $*$ yields a real-valued vector, each component representing the expected impact per time unit for each impact type.

For example, for a likelihood $l := 0.1/y$ ("expected to occur once every 10 years"), and an impact $i := (100\ \text{k€},\ 20\ \text{casualties},\ \text{legal consequences})$, the resulting risk can be obtained by

$$l * i = (10\ \text{k€}/y,\ \ 2\ \text{casualties}/y,\ \ \text{legal consequences every 10 years}).$$

Putting everything together, the risk associated to a single security event $\alpha \in V_\mathcal{G}$ can be determined as

$$\left(\sum_{v \in \text{rt}(V_\mathcal{G})} \mathbb{P}[v \rightsquigarrow \alpha] \cdot \mathbb{L}(v) \right) * \mathbb{I}(\alpha)$$

Note that by design, the latter expression covers exactly the risk coming from the event itself – not from the side effects it may possibly have. Therefore, the total risk of an entire risk scenario is precisely the sum of all eventually caused events. For a risk scenario described by an event σ, the total risk is thus

$$\sum_{\alpha \in \text{ante}(\sigma)} \left(\sum_{v \in \text{rt}(V_\mathcal{G})} \mathbb{P}[v \rightsquigarrow \alpha] \cdot \mathbb{L}(v) \right) * \mathbb{I}(\alpha), \tag{3.2}$$

where ante(σ) is the set of all security events that are eventually causing σ (or in graph language, the set of all antecedents of σ), including σ itself.

Similarly, the risk for the entire organisation can be computed as

$$\sum_{\alpha \in V_\mathcal{G}} \left(\sum_{v \in \text{rt}(V_\mathcal{G})} \mathbb{P}[v \rightsquigarrow \alpha] \cdot \mathbb{L}(v) \right) * \mathbb{I}(\alpha).$$

Note that in this sum, every security event is accounted for exactly once. In other words, it does not contain any redundant information.

3.2.5 Compatibility with standard risk methodologies

The work that was carried out during the thesis is supposed to be compatible with any risk management process, no matter the methodology. While this is an ambitious objective, and cannot be achieved for literally *all* methodologies, a decent amount of work has been put into the design of the framework, so that it can operate independently of the picked procedures.

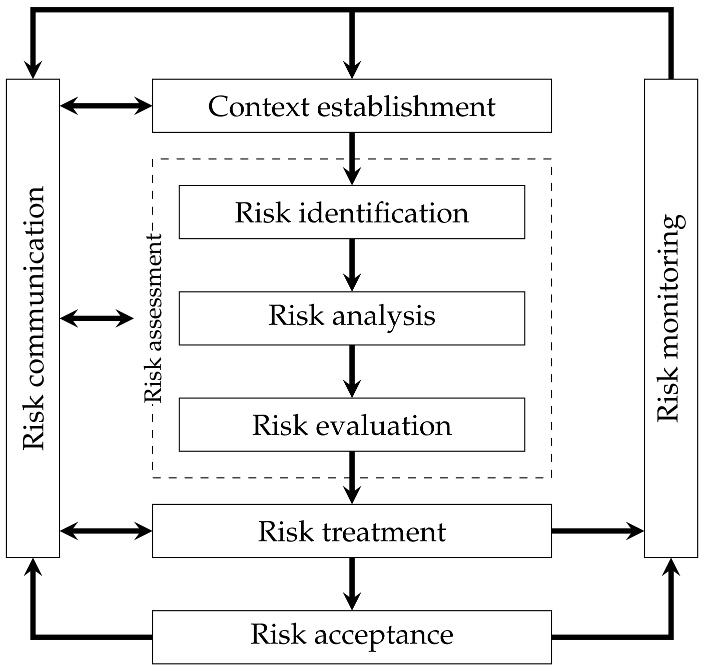


FIGURE 3.4 – The risk management process according to ISO 27005.

That is why the framework is presented as an *extension* to risk assessments, not a replacement thereof.

In fact, ISO 27005-compatible risk management processes are structured as depicted in Figure 3.4 [11]. Although other methodologies exist, they all require some form of risk assessment that is similar to the one of ISO 27005.

As is highlighted in Figure 3.5, the dependency model acts in parallel to the normal risk assessment process, even though it exchanges information with the latter. It additionally includes live data from the risk monitoring process into the dependency graph. In consequence, the risk assessment is no longer static and valid only for the moment when it was created, but it encodes a mean to automatically updates itself.

Note that all other processes are not affected by the additional features, and can be carried out just as usual.

A word on risk treatment. Although this thesis does not focus on risk treatment and mitigation, some hints are provided on how the dependency model can serve as decision criteria for implementing a security measure. Indeed, according to Section 3.2.1,

$$\text{risk} = \text{likelihood} \times (1 - \text{security}) \times \text{impact}.$$

The additional 'security' parameter accounts for the reduction in terms of risk; intuitively, it reflects the degree to which counter measures have been

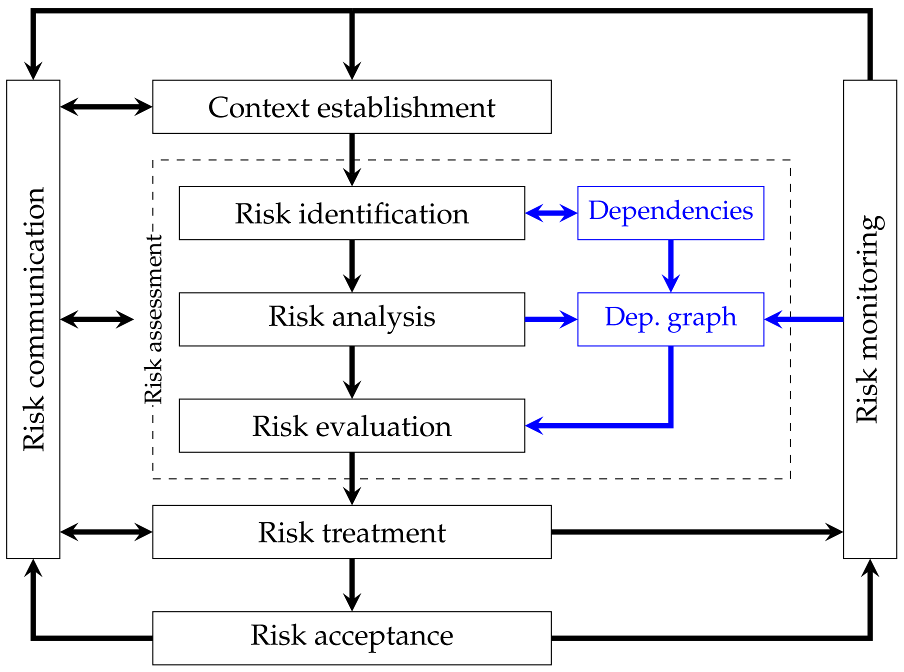


FIGURE 3.5 – Dependencies as an extension to the risk management process.

implemented. To some extent, the risk monitoring process could also 'measure' the effectiveness of these security controls and thus update the risk analysis (more precisely, the 'security' parameter) automatically – again, this is not within the scope of this thesis.

To determine the influence of counter measures, recall that the entire model is built around simulations of incidents. If one is to check whether a security control is worth being implemented, one can run the simulations twice – once with the control, and once without – and compare the decrease of risk with the associated costs. It is the risk acceptance criterion[3] that dictates how this comparison is made, and which risks shall be mitigated.

3.3 Computations

According to Equation 3.2 in Section 3.2.4, risk in a dependency graph is expressed in terms of the likelihoods $\mathbb{L}$, impacts $\mathbb{I}$, and the probability map p. While the first two parameters explicitly occur in the formulae, p hides in the expression $\mathbb{P}[\alpha \rightsquigarrow \beta]$, denoting the probability that event α eventually causes β. Unfortunately, computing this expression is non-trivial in general.

This section is thus dedicated to the numeric calculation of the probability distribution of causal relationships between events. To that end, an efficient algorithm is provided which can be used to update the involved probabilities in real time.

[3]The risk acceptance criterion is decided upon in the context establishment phase, before the analysis is actually conducted. Its form depends on the underlying risk methodology; for instance, in quantitative financial risk analyses, risk is typically accepted if the total costs of a security control outweigh its benefits, that is, its risk reduction.

3.3.1 Probability distribution of acyclic graphs

If the dependency graph is acyclic, it turns out that well-established theory can be used to mathematically express the full probability distribution. This case has been extensively studied [76], [77], [117]. It is important to note that already in this simpler case, it is computationally infeasible to determine the full probability distribution of general Bayesian networks [118]. Although some approximative algorithms exist [119], one needs to assume further properties on the graph.

A directed acyclic graph is said to satisfy the *local Markov property* [120] iff every node is independent from all of its non-descendants given its parents. Mathematically,

$$\forall v \in V \quad v \perp\!\!\!\perp \mathrm{nd}(v) \mid \mathrm{pa}(v),$$

where $\perp\!\!\!\perp$ denotes stochastic independence, $\mathrm{pa}(v)$ represents the set of immediate parent nodes of v, and $\mathrm{nd}(v) := \{v\} \cup (V \setminus \text{descendants}(v))$, the set of non-descendants.

Intuitively, this property requires that all causal relationships have been made explicit using edges. Note that this is the case if the dependencies encoded in the graph are exhaustive.

Denote by $\mathbb{I}_\alpha$ the indicator random variable[4] of a security event α. For readability, extend the notation to sets of events A as follows: $\mathbb{I}_A : \alpha \mapsto \mathbb{I}_\alpha$ for $\alpha \in A$. One can prove that for graphs satisfying the local Markov property, the following factorisation formula holds [121].

$$\mathbb{P}\left[\mathbb{I}_V\right] = \prod_{v \in V} \mathbb{P}\left[\mathbb{I}_v \mid \mathbb{I}_{\text{parents}(v)}\right].$$

By consequence, for any event $v \in V_{\mathcal{G}}$,

$$\begin{aligned} &\mathbb{P}\left[\mathbb{I}_v = 1 \mid \mathbb{I}_{\text{parents}(v)}\right] \\ &= 1 - \prod_{\substack{x \in \text{parents}(v) \\ \mathbb{I}_x = 1}} \left(1 - p(x, v)\right), \end{aligned} \tag{3.3}$$

which can be explicitly computed when $p : E_{\mathcal{G}} \to [0,1]$ is known. Moreover, for $\beta \in V_{\mathcal{G}}$,

$$\begin{aligned} &\mathbb{P}\left[\mathbb{I}_\beta = 1\right] \\ &= \sum_{\substack{\pi : V \to \{0,1\} \\ \pi(\beta) = 1}} \mathbb{P}\left[\mathbb{I}_V = \pi\right] \\ &= \sum_{\substack{\pi : V \to \{0,1\} \\ \pi(\beta) = 1}} \prod_{v \in V} \mathbb{P}\left[\mathbb{I}_v = \pi(v) \,\middle|\, \mathbb{I}_{\text{parents}(v)} = \pi|_{\text{parents}(v)}\right], \end{aligned} \tag{3.4}$$

[4] That is, $\mathbb{I}_\alpha = 1$ if α occurs and 0 otherwise.

where in the last step the factorisation formula [116] has been applied. It can even be shown[5] that it is enough to iterate over all ancestors of β (including β itself) instead of the whole vertex set V, which considerably reduces the computation effort.

Given a root event $\alpha \in V_{\mathcal{G}}$ (that is, one without parent nodes), one can condition on $\mathbb{I}_{\text{roots}}$ for both sides in (3.4) to get

$$\begin{aligned} & \mathbb{P}[\alpha \rightsquigarrow \beta] \\ = \; & \mathbb{P}\left[\mathbb{I}_\beta \mid \mathbb{I}_\alpha = 1, \mathbb{I}_{\text{roots}\setminus\{\alpha\}} = 0\right] \\ = \; & \sum_{\substack{\pi:V\to\{0,1\} \\ \pi(\beta)=1 \\ \pi(\alpha)=1 \\ \pi(\text{roots}\setminus\{\alpha\})=0}} \prod_{v\in V} \mathbb{P}\left[\mathbb{I}_v = \pi(v) \;\middle|\; \mathbb{I}_{\text{parents}(v)} = \pi|_{\text{parents}(v)}\right]. \end{aligned}$$

Plugging in the result from (3.3) yields the explicitly computable probability that an event β follows from α.

As can be observed in the final result, computing the expression requires iterating over all possible assignments $\pi : V \to \{0,1\}$ – thus the exponential running time.

3.3.2 Probability distribution for general graphs

For cyclically related security incidents, computing their likelihoods constitutes an even more delicate problem than it is already for acyclic ones.

In order to compute $\mathbb{P}[\alpha \rightsquigarrow \beta]$, one first needs to understand what it means to 'eventually trigger' an event. Consider two nodes $\alpha \neq \beta \in V_{\mathcal{G}}$. Recall that the dependency graph actually describes a simulation of incidents, so each edge represents a possible cascade effect (with the probability specified by p).

In the most basic case, where α and β are directly connected, and no other paths exist from α to β, the probability that α eventually causes β is just

$$\mathbb{P}[\alpha \rightsquigarrow \beta] = p(\alpha, \beta).$$

If a single path exists from α to β, with intermediate nodes $x_1, \ldots, x_{n-1}$,

$$\mathbb{P}[\alpha \rightsquigarrow \beta] = p(\alpha, x_1) \cdot p(x_1, x_2) \cdots p(x_{n-1}, \beta),$$

since all edges are assumed to be independent.

If multiple paths exist from α to β, the situation is not so trivial any more. For cyclic dependency graph, the issues becomes even more delicate.

However, from a probabilistic point of view, the dependency graph can also be seen as a random graph where each edge is present with a certain probability (the one specified by p). With that in mind, $\mathbb{P}[\alpha \rightsquigarrow \beta]$ is the probability that in a random sampling of the edges (according to p), there is a path from α to β.

[5]See Proposition A.1 in the appendix.

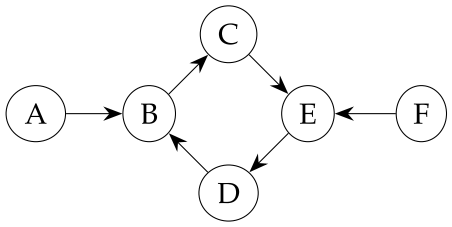


FIGURE 3.6 – Simple example of cyclically dependent events encoded in a graph.

Note how this perspective elegantly addresses the problem of cyclic dependencies. In fact, two events being cyclically dependent means exactly that each of them can (eventually) cause the other – but not more. In consequence, the fact that there is path from one to the other is all that matters – cycles don't change anything to that.

Remark 3.4. *Cyclic dependencies as introduced in this thesis cannot be used to account for recurrent events. The dependency graph only encodes causal relationships, and not actual behaviour over time. In other words, in the simulation of an incident, each event is either triggered or not – it will not be activated multiple times. In fact, likelihood is split up into two independent parts: a causal aspect (as specified by the graph) and a statistical aspect (encoded as the expected frequency* $\mathbb{L}$ *of the root causes), which should not be confused.*

Figure 3.6 explains the concept with the help of a simple dependency graph. Event E can be caused by C or F, but inspecting the situation in more detail, E is only caused by either of the two event chains $A \to B \to C \to E$ or $F \to E$. In particular, E can only by triggered by C if C is not already indirectly triggered by E (through the chain $F \to E \to D \to B \to C$).

As shown in this example, the probability that a risk scenario occurs cannot only be expressed by the probability of its direct parents, but has to involve all paths from a root node. Even worse, the computation effort for enumerating all such paths can be huge (in the worst case, namely in complete graphs, the running time is exponential in the number of vertices). That is also why any efforts of finding an efficient deterministic algorithm failed. In contrast, we managed to design a *probabilistic* algorithm that is able to yield a good approximation for the probabilities that shall be computed. Since the algorithm is randomised, its output is not deterministic and may be erroneous. However, we prove that the probability of returning a wrong output is so low that it can be neglected for all practical purposes.

3.3.3 Algorithm

Algorithm 1 makes use of the Monte Carlo method[7] to approximate the likelihoods. Its running time is polynomial in its input data. More precisely,

[6]Sampling a random graph consists in creating a new graph with the same nodes, and running a random experiment for each edge e, deciding whether it is included (with probability $p(e)$) or not.

[7]The Monte Carlo method consists in repeatedly sampling a probability distribution (here: random graphs) to obtain numeric results via statistical methods.

Algorithm 1 Compute probability matrix

```
Input: Graph $G = (V, E)$ with root nodes $V_R \subset V$
Input: Probability map $p : E \to [0, 1]$
Input: $\varepsilon > 0, \delta > 0$
Output: Probabilities $\mathcal{C} : V_R \times V \to [0, 1]$ that a root node causes a node,
   each value with absolute error at most $\varepsilon$. The algorithm will fail with
   probability at most $\delta$.
 1: ' Determine number of simulations $N$
 2: $\gamma := \frac{\varepsilon}{1+\sqrt{\varepsilon}}$
 3: $N := \frac{6}{\varepsilon^2\gamma} \ln\left(\frac{2n}{\delta}\right)$ where $n := |V|$
 4: for $(v_r, v) \in V_R \times V$ do
 5:     $\mathcal{C}(v_r, v) \leftarrow 0.$
 6: end for
 7: loop $N$ times
 8:     Sample[6]a random graph $G'$ from $G$ according to $p$
 9:     for $v_r \in V_R$ do
10:         for $v \in V(G')$ do
11:             if $\exists$ path in $G'$ from $v_r$ to $v$ then
12:                 $\mathcal{C}(v_r, v) \leftarrow \mathcal{C}(v_r, v) + 1/N$
13:             end if
14:         end for
15:     end for
16: end loop
```

it is bounded by

$$\mathcal{O}\left(n \cdot m \cdot \ln\left(\frac{2n}{\delta}\right) \cdot \varepsilon^{-3}\right),$$

where n is the number of vertices, m is the number of edges, δ is the probability that the algorithm output is wrong and ε is an upper bound for the absolute error of the computed values. Observe the logarithmic dependency on δ, which permits amplifying the algorithm accuracy without significantly increasing its running time.

The proof of correctness, running time and error probability is given in Proposition A.2 in the appendix.

3.3.4 Numerical experiments

While Proposition A.2 gives an upper bound of the running time *complexity*, one may also be interested in the actual time (in seconds) that it takes to execute the algorithm. This section presents the various numerical experiments that have been run in order to determine the dependence on the input parameters.

When inspecting the algorithm, one notices that it involves repeating the same random experiments over and over. Since all of these repetitions are carried out in an independent manner, they can be perfectly run in parallel.

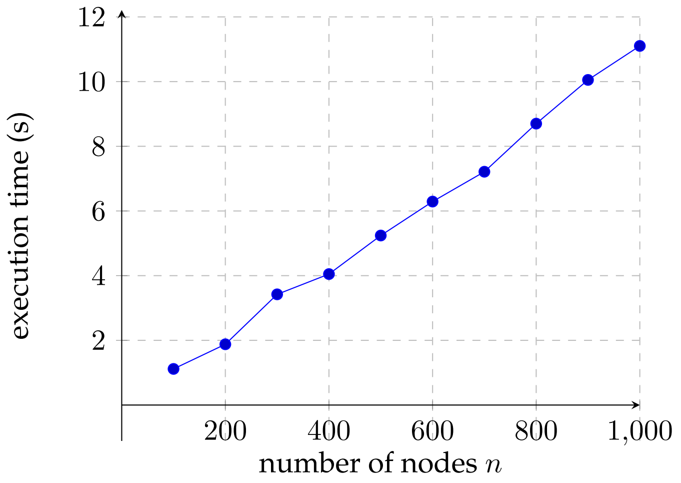


FIGURE 3.7 – Execution time of Algorithm 1 in seconds, depending on the graph size n, with $\varepsilon = 0.1$ and $\delta = 0.01$ and an average of 5 neighbours per node.

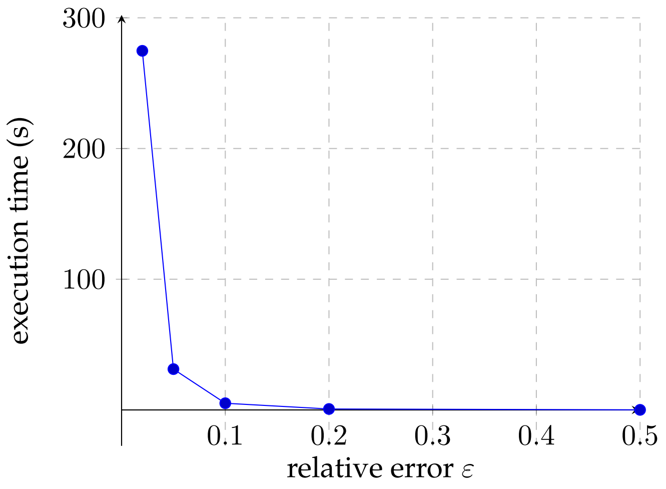


FIGURE 3.8 – Execution time of Algorithm 1 in seconds, depending on the precision ε of the results, with $n = 500$ and $\delta = 0.01$ and an average of 5 neighbours per node.

When doing so, one can profit from multi-threading capabilities of a CPU, or apply distributed computing techniques.

To test the performance of the algorithm on 'average' graphs, dependency graphs have been generated uniformly at random. A typical risk analysis may cover up to 50 different assets[8], each of which generally encounters 3–5 threats[9], so a related graph is composed of a few hundred nodes. It is sensible to assume that nodes are not connected (in average) to more than a few edges, so a typical graph will consist of a few thousand edges at most.

The simulation was performed on a dual-core 2.5 GHz processor (i7-3537U). The results are depicted in Figures 3.7, 3.8, 3.9 and 3.10 – as expected, they

[8]Based on experience from past risk analyses performed by *itrust consulting*.

[9]Most often, these threats include the criticality, integrity and availability aspects of each asset, which can be further sub-divided (e.g. temporary unavailability vs. permanent loss).

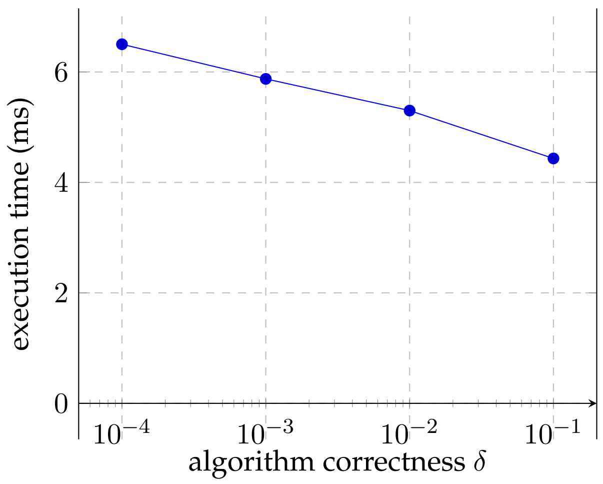


FIGURE 3.9 – Execution time of Algorithm 1 in seconds, depending on the correctness δ of the algorithm output, with $n = 500$ and $\varepsilon = 0.1$ and an average of 5 neighbours per node.

reflect the running time computed in Proposition A.2 in the appendix. The precise numbers can be found in Table B.1 in the appendix.

In order to compare the performance of Algorithm 1 to other approaches, similar experiments have been conducted with for straight-forward deterministic algorithms. For example, consider the simple recursive algorithm which conditions on the existence of each edge. The former relies on the mathematical observation that, for each edge $e \in E$,

$$\begin{aligned}\Pr[\exists \text{ path } v \rightsquigarrow w] = & \qquad p(e)\cdot \Pr[\exists \text{ path } v \rightsquigarrow w \mid e] \\ & +(1-p(e))\cdot \Pr[\exists \text{ path } v \rightsquigarrow w \mid \neg\, e],\end{aligned}$$

which can easily be turned into a recursive algorithm, computing the probability that a path exists between any two nodes v and w. Unfortunately, such algorithms have exponential running time and take more than a few minutes already for small graphs ($|V| \geq 20$, $|E| \geq 200$).

All other attempts to solving the problem in a deterministic way resulted in similarly bad execution times.

3.4 Sensitivity

It is commonly the case that some information is incomplete or unavailable to the risk assessor when he sets up the risk context. In that case, he is required to estimate or guess some of the parameters, such as the likelihood or impact. While for traditional risk analyses, errors in these estimates only influence a single risk scenario, dependencies can cause the error to spread through the entire analysis.

Moreover, since automated processes yield values that are not necessarily verified by human experts, it is also important to understand how the model reacts to changes in the input.

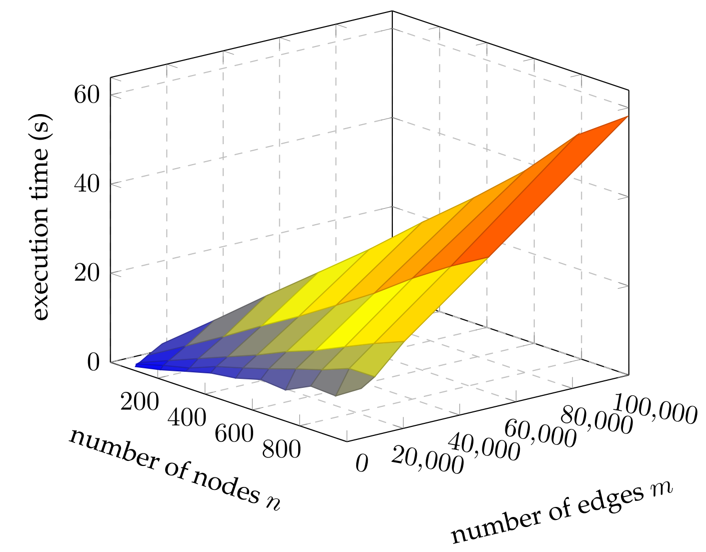


FIGURE 3.10 – Execution time of Algorithm 1 in seconds, depending on the graph size n and m, with $\varepsilon = 0.1$ and $\delta = 0.01$.

This section is dedicated to analysing the extent to which the dependency model is sensitive to uncertainties in the input data and to give bounds on the engendered forward error.

3.4.1 Aborting computation prematurely

Algorithm 1 essentially works by repeating the simulation of a random experiment, and by taking the average outcome. Although this way of proceeding only yields approximative results, Proposition A.2 proves that the obtained values are precise enough for everyday use, if the experiment is only run often enough.

In fact, the more often the experiment is repeated, the more precise the output will get. In particular, the algorithm can be interrupted at any time, in which case it yields a complete, but less precise solution. Since for large risk analyses, the running time can have an order of magnitude of several seconds or even minutes, one may be tempted to stop the computations prematurely. For real-time applications, such as risk monitoring, one may even be *required* to abort the algorithm.

In that case it is important to understand the meaningfulness of the (incomplete) results. Lemma 3.5 below address this question by proving an upper bound for the relative error of the computed values.

Lemma 3.5. *If Algorithm 1 is aborted after αN iterations, for $0 < \alpha < 1$, then the relative error of the algorithm output increases at most by a factor $\alpha^{-\frac{1}{3}}$.*

Proof. By definition, the algorithm is run $N_0 := \frac{1+\sqrt{\varepsilon_0}}{\varepsilon_0^3} \cdot 6 \ln\left(\frac{2n}{\delta}\right)$ times, for a previously fixed relative error ε_0. If instead of ε_0, one would pick a relative error of $\varepsilon := \beta\varepsilon_0$, it would require

$$\begin{aligned} &\frac{(1+\sqrt{\varepsilon})}{\varepsilon^3} \cdot 6 \ln \frac{2n}{\delta} \\ =\ &\frac{(1+\sqrt{\beta}\sqrt{\varepsilon_0})}{\beta^3 \varepsilon_0^3} \cdot 6 \ln \frac{2n}{\delta} \\ =\ &\beta^{-3} \frac{(1+\sqrt{\beta}\sqrt{\varepsilon_0})}{(1+\sqrt{\varepsilon_0})} \cdot N_0 \\ \leq\ &\beta^{-3} \cdot N_0 \end{aligned}$$

iterations instead, for any $0 < \beta < 1$.

Since the outputs get preciser the longer the algorithm runs, running it precisely $\beta^{-3} N_0$ times will yield a relative error $\beta\varepsilon_0$ (or even better). Setting $\beta = \alpha^{-\frac{1}{3}}$ concludes the proof. $\square$

For instance, aborting the algorithm after half of the required steps ($\alpha = \frac{1}{2}$) would cause the relative error to increase by

$$\left(\frac{1}{2}\right)^{-\frac{1}{3}} - 100\% = \sqrt[3]{2} - 100\% \approx 26\%,$$

which is (of course) less precise, but still within the same order of magnitude.

3.4.2 Varying graph topology

When building the graph, risk assessors need to be as concise as possible. In particular, they should avoid encoding *indirect* relationships $A \to C$ if A actually causes C indirectly through B (i.e., $A \to B \to C$). Nevertheless, it is sometimes not so easy to know if there is a direct or indirect relationship between two events. Since errors in that respect are to be expected, it is important to analyse the influence of the graph topology on the obtained probabilities.

Lemma 3.6 below gives a rather pessimistic bound on the impact of an additional edge in the graph. It essentially states that adding edges can only increase the probabilities $\mathbb{P}$, but that this increase is also bounded by the probability of the edge itself. If, however, the latter probability is large (e.g., 0.9 or even 1), then this statement is no longer useful.

Lemma 3.6. *Let $G = (V, E)$ be a dependency graph, and $G' = (V, E')$ be the same graph with one (any) additional edge e. Then for any two nodes $\alpha, \beta \in V$,*

$$\mathbb{P}[\alpha \rightsquigarrow_G \beta] \quad \leq \quad \mathbb{P}[\alpha \rightsquigarrow_{G'} \beta] \quad \leq \quad \mathbb{P}[\alpha \rightsquigarrow_G \beta] \cdot (1 - p_{G'}(e)) + p_{G'}(e).$$

Proof. The first inequality follows readily from the fact that additional edges can only increase probabilities. The second inequality can be inferred from conditioning on the presence of the edge e:

$$\begin{aligned}\mathbb{P}[\alpha \rightsquigarrow_{G'} \beta] = &\quad \mathbb{P}[\alpha \rightsquigarrow_{G'} \beta \mid \neg e] \cdot (1 - p_{G'}(e)) + \mathbb{P}[\alpha \rightsquigarrow_{G'} \beta \mid e] \cdot p_{G'}(e) \\ \leq &\quad \mathbb{P}[\alpha \rightsquigarrow_{G} \beta] \cdot (1 - p_{G'}(e)) + p_{G'}(e)\end{aligned}$$

since $\mathbb{P}[\alpha \rightsquigarrow_{G'} \beta \mid \neg e] = \mathbb{P}[\alpha \rightsquigarrow_{G} \beta]$ by assumption on G and G'. □

Instead of analysing the situation for a general graph, one can come back to the original problem. However, even for triangular relationships of the form $A \to B \to C, A \to C$ (that correspond exactly to the direct/indirect relationship problem mentioned above), we did not manage to find a better estimate for the above.

3.4.3 Varying edge probabilities

Once the graph structure is fixed, the next step consists in determining the edge probabilities. As the previous process, this task is subject to arbitrariness of the risk assessor. Indeed, there is no intuitive difference between a probability of 75% and 70%. However, Lemma 3.7 states that small changes in the input data also reflect in small changes in the output data. In other words, small errors will not be unexpectedly strengthened and create huge deviations in the output.

Lemma 3.7. *Let $G = (V, E, p)$ be a dependency graph with probability map p. Let $e \in E$ be any edge. Consider the graph $G' = (V, E, p')$ that entirely corresponds to G except for $p'(e) \neq p(e)$. Then for any $\alpha, \beta \in V$,*

$$|\mathbb{P}[\alpha \rightsquigarrow_{G} \beta] - \mathbb{P}[\alpha \rightsquigarrow_{G'} \beta]| \leq |p(e) - p'(e)|.$$

Proof. Note that G and G' only differ by the probability value for e. Therefore, when conditioning on the presence of e, it actually holds that

$$\mathbb{P}[\alpha \rightsquigarrow_{G'} \beta \mid e] = \mathbb{P}[\alpha \rightsquigarrow_{G} \beta \mid e],$$

and similarly for $\mathbb{P}[\alpha \rightsquigarrow_{G'} \beta \mid \neg e]$. Moreover,

$$\begin{aligned}\mathbb{P}[\alpha \rightsquigarrow_{G} \beta] = &\quad \mathbb{P}[\alpha \rightsquigarrow_{G} \beta \mid e] \cdot p(e) + \mathbb{P}[\alpha \rightsquigarrow_{G} \beta \mid \neg e] \cdot (1 - p(e)) \\ = &\quad c_1 \cdot p(e) + c_0 \cdot (1 - p(e)),\end{aligned}$$

for constants $0 \leq c_0, c_1 \leq 1$. Similarly for G',

$$\mathbb{P}[\alpha \rightsquigarrow_{G'} \beta] = \quad c_1 \cdot p'(e) + c_0 \cdot (1 - p'(e)).$$

The proof follows from

$$\begin{aligned}\mathbb{P}[\alpha \rightsquigarrow_{G} \beta] - \mathbb{P}[\alpha \rightsquigarrow_{G'} \beta] = &\quad c_1 \cdot (p(e) - p'(e)) - c_0 \cdot (p(e) - p'(e)) \\ = &\quad (c_1 - c_0) \cdot (p(e) - p'(e))\end{aligned}$$

and from the fact that $|c_1 - c_0| \leq 1$ □

In extreme cases, an error of 10% for the input will also result in an error of 10% for the output. It is, for example, the case for the very simply dependency graph consisting of only $A \to B$.

However, in practise, graphs are rather complex. In fact, the more paths exist between two nodes, the less influence estimation errors will have on the overall outcome. In other words, the error estimated by Lemma 3.7 will be even smaller.

3.4.4 Varying likelihoods and impacts

A final question addresses how uncertainties in the likelihood or impact estimations of risk scenarios reflect in the computed risk. Whereas the two previous sections deal with the graph structure, this issue only concerns the risk formula (3.2):

$$\sum_{\alpha \in \text{ante}(\sigma)} \left(\sum_{v \in \text{rt}(V_{\mathcal{G}})} \mathbb{P}[v \rightsquigarrow \alpha] \cdot \mathbb{L}(v) \right) * \mathbb{I}(\alpha).$$

It is to be observed that the risk formula above depends bilinearly on the likelihoods and impacts of (a selection of) risk scenarios. Uncertainties in the input ($\mathbb{I}$, $\mathbb{L}$) thus reflect linearly in the output.

Furthermore, the nature of the bilinear coefficient (which happens to be a probability, and thus bounded by 1), guarantees that these uncertainties appear in the risk only to a limited extent. For example, if the likelihood is assessed with an error of $\pm 0.1/y$, then the resulting risk has an error of at most $\pm 0.1\mathbb{I}/y$.

Lastly, the influence of a single error is reduced even further as the dependency graph grows. Indeed, since the summation in (3.2) goes over all $\alpha \in \text{ante}(\sigma)$, each term contributes to the total sum only to a limited extent. In fact, if the weightings are of the same order of magnitude (as recommended in Section 3.2.3), and dependency graphs are not too extreme (in the sense that all causal chains are more or less equally long), then the probabilities $\mathbb{P}[\alpha \rightsquigarrow \beta]$ are also of the same order of magnitude. In consequence, each term in (3.2) contributes roughly $\frac{1}{|\text{ante}(\sigma)|}$ to the total risk. The larger the graph, the larger the set of antecedents (causes), and thus the smaller the influence of errors in the output.

3.5 Building dependency graphs

When confronted with the task of creating a dependency graph, risk assessors might not know where to start. The objective of this section is to provide guidance on identifying relevant assets, intermediate security events, and finally the causal relationships that makes up such a dependency model.

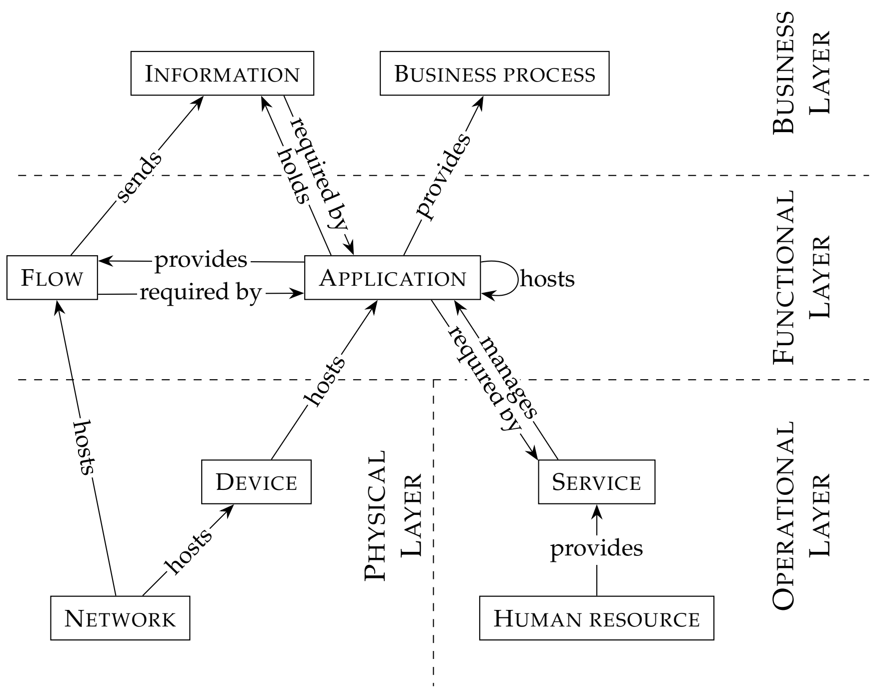


FIGURE 3.11 – Class diagram representing the taxonomy of assets involved in a risk analysis, grouped by layer. Read an edge '$c_1 \xrightarrow{\text{predicate}} c_2$' as '$c_1$ predicate c_2'.

To start, a taxonomy is presented that defines a systematic approach of going through the scope of a risk analysis, so as to determine all relevant assets for the dependency graph.

3.5.1 Taxonomy

Most (yet not all) security events are related to a physical or digital asset. For this reason, it makes sense to group assets facing similar threats, so as to express the dependencies between *asset classes* as causal relations between *threats* acting upon them. For instance, the natural dependence of sensitive data on the confidentiality of its database is contained in the statement that threats to any software also put any data at risk, that is managed by the latter.

The following asset classes have been identified. Figure 3.11 shows the same asset types together with their relationships in terms of risk; as for the dependency model presented in Section 3.2, an edge $\alpha \rightarrow \beta$ denotes that an incident in α can cause another incident in β.

- A *Service* is an internal or external supporting or maintenance unit. It is typically in charge of the well-functioning of the internal processes of the organisation. Examples include IT, HR, physical security, or external dependencies such as internet service providers and subcontractors.
- A *Human resource* is any individual, group of individual, or job position that is particular interest for an organisation. Examples include

system administrators, members of management staff (CxO's), heads of department, etc.

- A *Network* is a closed environment of interconnected devices. Communication to the outside is possible, but subject to rules (often imposed by a firewall). Compromising a network amounts to compromising the flows provided by devices inside that network.
- A *Device* is any physical hardware. Devices are subject to mechanical damage and physical access.
- An *Application* is the functional (software) counterpart of a device. Unlike a device, an application is threatened by software vulnerabilities and remote attacks. The idea is to separate the soft- and hardware layer, since both are exposed to different risk scenarios.
- A *Flow* transports data from one application to another over a network. Flows can be manipulated/blocked and tapped.
- *Information* comprises all kind of data, including raw data contained in a database, keys, passwords and certificates. Information can be destroyed, be tampered with or leak.
- A *Business process* consists, very abstractly, in a goal that an organisation is pursuing. It can represent a department, a service that is provided to externals (e.g., customers), compliance, or similar.

When designing a dependency graph, this taxonomy helps in determining the assets in a exhaustive way. It is recommended to start at the business layer and move on downwards in the taxonomy, following the relationships depicted in Figure 3.11. The procedure is the following.

1. For each asset class, risk assessors should first create a list of all related assets that are relevant for a risk analysis, i.e., which are important to the organisation in questions, or have risks associated to them.
2. In a second step, possible security events should be determined for each of the identified assets. A security event can be any fault, issue, or problem that occurs for the asset in question, which is part of or can give rise to a risk scenario. In order to systematically find all relevant security events, one can go through the security properties of an asset (such as confidentiality, integrity, availability), and determine the consequences when they are missing. Also see Table 3.2 for a list of security aspects that are commonly considered.
3. Finally, the security events for each of these assets should be encoded in the dependency graph, and be connected to their respective causes via edges. As defined in Section 3.2, an edge $\alpha \to \beta$ denotes the fact that an event α may give rise to another event β.

Step 2 in the previous procedure can also be omitted, so that the dependency graph contains assets as its nodes (instead of security events) and asset dependencies as its edges (instead of causal relationships).

In such a case, the model can still be interpreted as a normal dependency graph, if one equates a node to the totality of security incidents that can

Symbol	Property	Scenarios
A_{lt}	Long-term availability	Permanent loss, deletion, theft, or destruction. Can be caused accidentally or intentionally.
A_{st}	Short-term availability	Temporary unavailability, downtime (hard-/software), or sickness (human resources).
C_1	Secrecy	Theft or disclosure of secret information. May result in fraudulent access.
C_*	Confidentiality	Public disclosure of confidential information (such as personal data). Often has impact on privacy. Can be accidental or intentional.
I_{in}	Intrinsic integrity	Dysfunction, bug (hard-/software), data corruption, dishonesty or corruption (human resources).
I_{ex}	External integrity	Manipulation, intrusion (systems and networks), social engineering (human resources). Can originate from internal staff or from externals.

TABLE 3.2 – Common security aspects that help in determining security events.

possibly occur to the asset represented by the node. While the pure asset dependency graph constitutes an oversimplified variant of the original model, it still excels by its simplicity and the small amount of work required to elaborate it.

3.5.2 DepOT

To demonstrate the compatibility of the model with casual risk methodologies, a small tool ('DepOT') has been developed in the context of the thesis, which synchronises a dependency graph with TRICK[10], an ISO 27005 compliant risk methodology.

The web-based tool is written entirely in Javascript and HTML5, and provides drag-and-drop functionality to come up with a dependency graph from nothing. Figure 3.12 shows a screenshot of the user interface.

In addition to saving and loading the graph to and from a JSON file, the dependency graph can also be linked with the risk analysis tool 'TRICK Service'. That way, an existent risk analysis (that lacks dependencies) can be imported into DepOT, which enables risk assessors to easily model the dependency graph. If the risk analysis managed by 'TRICK Service' is updated (e.g., new assets are added), the tool can also make these changes reflect in the associated dependency graphs.

The DepOT tool is an example of a purely manual method to create a dependency graph. Especially for large organisations, however, building such

[10]https://www.itrust.lu/trick-service/

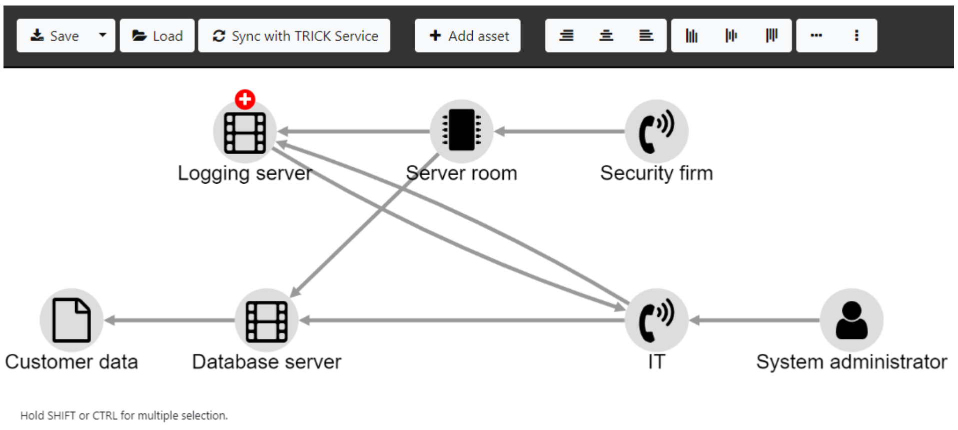


FIGURE 3.12 – User interface of the DepOT tool.

a graph is tedious and repetitive, since many assets share the same threats. To overcome the problem of redundant work, the next section presents a semi-automated approach that generates a dependency graph from a small set of rules (which still need to be elaborated manually).

3.5.3 Semi-automated generation

Dependency graphs can improve the readability of a risk analysis tremendously, since they visually represent how assets or scenarios are linked together. However, as the graph grows in terms of nodes and edges, it also loses some of its legibility. Figure 3.13 shows an example of a graph which consists of so many nodes that the readability is entirely lost. Even worse, since the probability map p needs to be manually estimated by an expert, a large graph requires a considerable estimation overhead.

The fact that dependency graphs can quickly become large and complicated constitutes an issue that needs to be addressed. Fortunately, the size of the graph is often due to the fact that many assets share the same threats, and thus give rise to a certain information redundancy in the graph. For instance, an organisation may have several servers that serve different purposes, but all of them feature the risk of prolonged downtime.

Instead of encoding the risk scenario for each server individually, a more clever approach would be to encode it only once (for servers in general), and copy the definition for each server identified in the risk analysis. The main idea is then to use the taxonomy presented above to express *risk* dependencies in terms of *asset* dependencies. For example, instead of encoding the dependency

$$\text{database server crash} \rightarrow \text{data loss},$$

the taxonomy specifies that

$$\forall\,\text{server } S, \quad S \text{ crash} \rightarrow \text{loss of (data stored on } S\text{)}.$$

So, in particular it is true for $S = $database server.

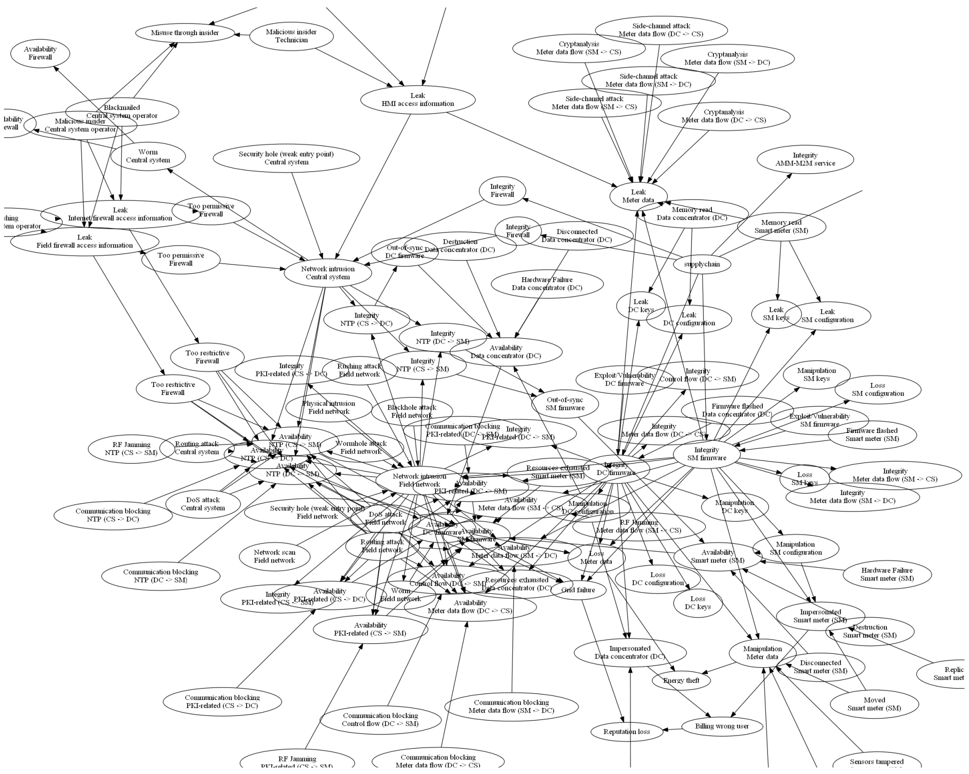


FIGURE 3.13 – Example of an overfull dependency graph.

Intuitively, one can think of the taxonomy as a template for dependency graphs (as introduced in Section 3.2 above), that can be applied to all assets of a specific type.

These risk dependency 'templates' are encoded in a simple mark-up language that is based on the GraphViz[11] DOT syntax. They are expressed as

"$node_A$" -> "$node_B$" [p = x];

which reads as

If $node_A$ occurs, it causes $node_B$ to occur with probability x;

where $node_A$ and $node_B$ are the IDs of the respective nodes in the dependency graph, and $x \in [0, 1]$. The syntax is extended in such a way that the node IDs can contain placeholders which match whole asset classes. Placeholders are enclosed with angular brackets < and > and are of the following form.

< *assetclass* >
< *assetclass* . *selector* . *selector ...* >
< *assetclass* # *filter* . *selector ...* >
< *assetclass* # *filter* . *selector* # *filter ...* >

where

- *assetclass* is one of `net`, `dev`, `hr`, `svc`, `app`, `flow`, `inf`, `pro`;

[11]GraphViz is an open-source graph visualization software. For more information on the DOT language, see http://graphviz.org/content/dot-language.

- *selector* is one of these: `held-inf`, `hosted-app`, `hosted-dev`, `hosted-flow`, `managed-app`, `provided-flow`, `provided-pro`, `provided-svc`, `requiring-app`, `requiring-svc`, `sent-inf`, depending on the context. The selector serves as place holder for dependent assets, and thus allows to navigate through the class model depicted in Figure 3.11;
- *filter* is a keyword restricting the choice of selected nodes (e.g. `#fw` for only selecting firewall devices, `#key` for only selecting encryption keys).

Moreover, if there are placeholders on both sides of the '`->`', the asset class on the right hand-side shall match the one on the left hand-side.

For instance, the following line encodes the fact that if an attacker has full control over any application, there is a 10% chance that he gets access to any associated secret keys.

```
"<app>.control" -> "<app.held-inf#key>.leak" [p = 0.1];
```

In order to be able to deduce the final dependency graph from the definitions, an inventory describing the assets themselves needs to be created. Note that organisations usually have such an inventory, especially if they have a security management systems. The inventory itself can be encoded as a directed graph in the DOT syntax as well.

```
    # Defining assets
    "inf:cdat" [label = "Customer data"];
    "app:db" [label = "Database"];

    # Defining relations
    "app:db" -> "inf:cdat" [label~=~"held-inf"];
```

Node IDs should be prefixed by the asset class (e.g. `inf:` for information assets) so that the class can be inferred from the ID. Edge labels show the kind of relation which the first (left) node maintains to the second (right). The relation is expressed in terms of one of the selectors from the list above.

Once the dependencies have been defined and the inventory has been established, both inputs can be programmatically combined to yield the dependency graph – the approach is described in Algorithms 2 and 3.

3.6 The 'Smart Grid Luxembourg' use-case

The 'Smart Grid Luxembourg' (SGL) project aims at innovating the electricity and gas transmission networks by deploying a nation-wide 'smart' grid infrastructure. By law, starting from July 2016, every new electricity or gas meter deployed in Luxembourg will be a smart meter. At the end of the project, by 2020, the system will count 380.000 smart meters. In the following, the concepts presented in Section 3.5.3 are applied to SGL.

Algorithm 2 Generate dependency graph from template and inventories

Input: Template dependency graph $T = (V_T, E_T, p_T)$
Input: Inventory graph $I = (V_I, E_I, s_I)$, where $s_I(e)$ denotes the *selector* represented by an edge e (defined in Section 3.5.3)
Output: Dependency graph $G = (V, E, p)$

```
 1: $V \leftarrow \emptyset$
 2: for $v_t \in V_T$ do
 3:     $V \leftarrow V \cup \text{resolve}(v_t, I, \textbf{nil})$
 4: end for
 5: $E \leftarrow \emptyset$
 6: $p : V \times V \rightarrow [0,1], (u,v) \mapsto 0$
 7: for $e_t = (u_t, v_t) \in E_T$ do
 8:     for $u \in \text{resolve}(u_t, I, \textbf{nil})$ do
 9:         for $v \in \text{resolve}(v_t, I, u)$ do
10:             $E \leftarrow E \cup \{(u,v)\}$
11:             $p(u,v) \leftarrow p_T(u_t, v_t)$
12:         end for
13:     end for
14: end for
15: return $(V, E, p)$
```

Algorithm 3 resolve(v_t, I, c) used in Algorithm 2: Resolve a template node to a set of nodes in the final dependency graph.

Input: Template node $v_t \in V_T$ that shall be resolved
Input: Inventory graph $I = (V_I, E_I, s_I)$ as in Algorithm 2
Input: Context node $c \in V$ or **nil**
Output: Set of resolved nodes $\mathcal{R}$ in the dependency graph

```
 1: /* For starters, get all nodes that match the asset class */
 2: $S \leftarrow \{x \in V_I \mid \text{assetclass}(x) = \text{assetclass}(v_t)\}$
 3: if $c$ is not nil then
 4:     $S \leftarrow S \cap \{c\}$
 5: end if
 6: $\mathcal{R} \leftarrow S$
 7: /* Navigate through the model by processing selectors one-by-one */
 8: for $s \in \text{selectors}(v_t)$ do
 9:     $S \leftarrow \emptyset$
10:     /* Apply selector to each of the found nodes */
11:     for $r \in \mathcal{R}$ do
12:         $S \leftarrow S \cup \{x \in V_I \mid s_I(r,x) = s\}$
13:     end for
14:     $\mathcal{R} \leftarrow S$
15: end for
16: return $\mathcal{R}$
```

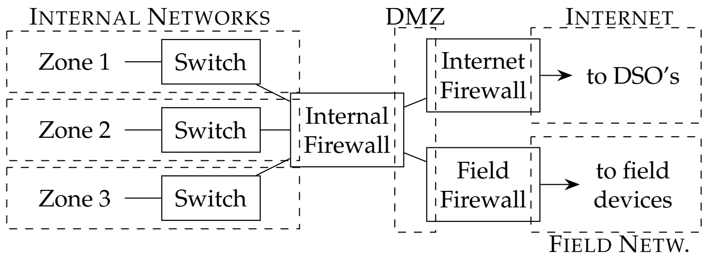


FIGURE 3.14 – Anonymised network diagram of the central system architecture showing devices and the their affinity to the respective networks. *DSO* denotes a *Distribution System Operator*; *DMZ* stands for *DeMilitarised Zone*; field devices include data concentrators and smart meters.

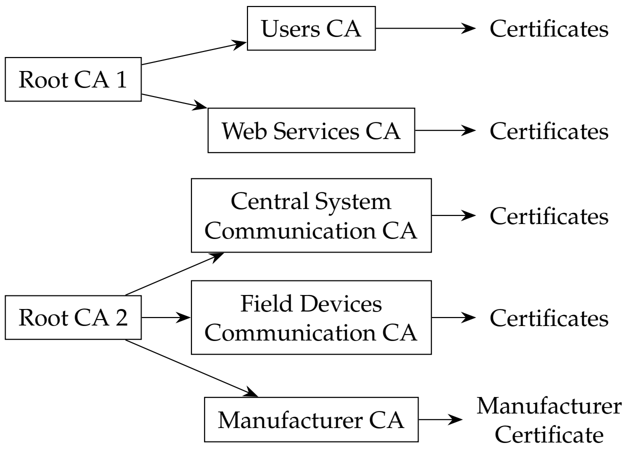


FIGURE 3.15 – Anonymised hierarchy of certificates used in the smart grid. *CA* denotes a *Certificate Authority*.

3.6.1 Compiling a dependency-aware inventory

In a first phase, the inventory of all relevant assets has been compiled, covering hard- and software, physical wiring, network flows, database tables and their contents, certificates, other kinds of information and the services provided by the various applications. Figures 3.14, 3.15 and 3.16 provide anonymised variants of the complete, confidential graphs.

The Luxembourgish smart grid manages its own Public Key Infrastructure (PKI), so as to guarantee complete independence of any external providers. The certificates in a PKI bear a natural dependency hierarchy with them, in the sense that compromising any certificate authority (CA) permits to reproduce any dependent certificates and thus, ultimately, to undermine an encrypted communication channel.

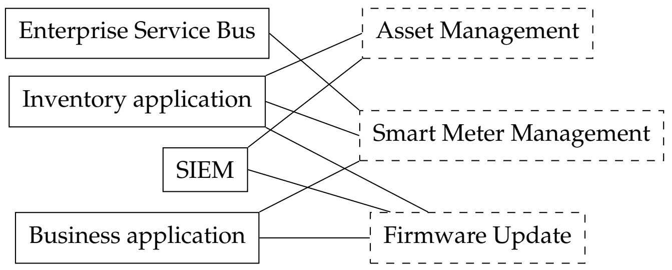


FIGURE 3.16 – Excerpt of the matching between applications (solid boxes) and services (dashed boxes). *SIEM* denotes the *Security Information and Event Management* appliance.

3.6.2 Threat model

The second phase consisted in identifying all possible threats faced by the system and encoding them properly in a dependency graph (with placeholders).

The elaborated threat model is based, on the one hand, on other research work by Grochocki et al. [122] and ENISA [123], who determine the threats faced by a smart grid infrastructure. On the other hand, the 'Smart Grid Luxembourg'-specific dependencies could be extracted from former risk analyses and from documentation material that was kindly provided by Luxmetering.

It turns out that a large portion of the threats can be expressed as a tuple consisting of an asset (class) and an endangered security property (such as confidentiality, integrity or availability). For instance, the generic risk scenarios faced by applications comprehend malfunctioning, unauthorized access (e.g. by faking login credentials), lose of control (e.g. due to code injection) and denial of service. The remaining risk scenarios, which are not directly associated to an asset (such as distributed denial-of-service or fire incident), are added as singleton nodes to the graph.

3.6.3 Generating the dependency graph

Once the threat model was set up, the final dependency graph could be programmatically derived from the inventory. For this purpose, a *Python* implementation of Algorithm 2 reads in the dependency definitions and applies them to the inventory (by replacing all placeholders by the respective IDs of the assets in the inventory).

Algorithm 1 permits then to identify all root causes, that is to say, to find those events which are ultimately responsible for all risk scenarios in the threat model. Moreover, it determines the probabilities that *each* of these root causes eventually leads to *each* of the other events by cascading effect – thus computing the probabilities $\mathbb{P}[\alpha \to \beta]$ involved in Equation 3.2 for determining the total risk.

3.6.4 Results

The inventory consists of 12 different devices (each with multiplicity[12]), 9 networks, 37 applications, 43 flows, 14 data sets, 26 certificates, 9 sets of credentials and 18 services. The generic dependency graph encoding the threat model (with placeholders) has 53 nodes and 104 edges. The time needed for conducting the risk analysis for Luxmetering is composed as follows:

Gather asset inventory from documentation material and past (static) risk analyses	18 h (2 md)
Define (generic) dependency graph	30 h (4 md)
Estimate $\mathcal{P}$ and $\mathcal{L}$	9 h (1 md)
Fine-tuning of the model	7 h (1 md)
Total	**64 h (8 man-days)**

Since the generic dependency graph contains (almost) no SGL-specific information, it can be easily recycled for other, similar use-cases.

Computing the full final dependency graph (consisting of 502 nodes and 1516 edges) took 3.79 seconds on a 2.0 GHz dual-core processor (which includes the parsing time of the inventory files). The probabilities $\mathbb{P}[\alpha \to \beta]$ have been computed using a C# implementation of Algorithm 1 (the chosen parameters were $\varepsilon = 0.1$ and $\delta = 0.01$); it is composed of 502 rows (as many as nodes) and 25 columns (root causes), comprising thus a total of 12550 probability values. Its computation took 39.14 seconds (which is 3.11 ms per value) on the same machine.

The following 25 root causes were read off the model:

- phishing, social engineering,
- bad input validation, XSS, CSRF, broken authentication, buffer overflow,
- DDoS, jamming, smart meter intrusion, physical access to facility,
- data center incidents (fire ...), device construction faults, mechanical attrition
- and 11 SGL-specific attacks.

The most critical risks identified by the algorithm were the following:

- manipulation of billing data,
- disclosure of customer data, requiring reporting to authorities and informing customers,
- power outages,
- forensics,
- loss of smart meter configuration data, which involves reconfiguring all 380.000 devices, and

[12]Some devices exist in redundant fashion (switches, servers) or in copies (smart meters).

- loss of billing data.

In total, a yearly risk of an order of magnitude of 300 k€ was estimated. The detailed risk analysis cannot be published, though, for confidentiality reasons.

3.7 Extensions and special cases

3.7.1 Boolean formulae

The dependency graph is based on the concept of causality; that is, the parents of a node represent alternative causes, each of which can engender the consequential scenario. Formally, the dependency relationship of a vertex $v_0 \in V_\mathcal{G}$ and its parent nodes $P_{v_0} \subset V_\mathcal{G}$ can be expressed as a boolean formula

$$\rho(v_0) := \bigvee_{x \in P_{v_0}} \mathbb{I}_x,$$

where $\mathbb{I}_x$ denotes the boolean variable encoding whether the event x occurs or not.

The beauty of Algorithm 1 lies in the fact that it does not depend at all on the topology of the graph or on the form of the dependencies. In fact, generalising the 'OR' relations to arbitrary boolean expressions $\rho(\cdot)$ is straightforward and does not change the main lines nor the proof of the algorithm.

In contrast, the running time will increase by much, in general. Intuitively, it is easy to determine whether a parent node triggers a child, for it is enough to verify if the corresponding edge is present. However, for arbitrary boolean formulae, more advanced theory (namely boolean satisfiability [124]) is required to decide whether a node is caused by its antecedents, or not. In the worst case, the whole graph needs to be evaluated, which may take a long time.

More precisely, the comparatively good running time of Algorithm 1 was due to the fact that evaluating the probabilities $\mathbb{P}[\alpha \rightsquigarrow \beta]$ can be implemented in an efficient way (line 10 of Algorithm 1). In fact, it is enough to verify that the path from α to β is present. The underlying problem[13] for the boolean-formula variant, however, is $\mathcal{NP}$-complete [124], which means that there does not exist an efficient algorithm. In consequence, even deterministic (and error-free) algorithms could outperform the algorithm presented in this thesis, which renders the latter useless.

Moreover, a second issue needs to be taken care of. In fact, it may be the case that a recursive search is no longer possible; for example, Figure 3.17 shows an endless loop in the evaluation of the boolean formula, which cannot be easily resolved using logic. In fact, in order to evaluate $A \wedge X$, one needs to evaluate *both* parents, including X and thus $Y \wedge B$ and Y. But Y can only

[13]Given an arbitrary boolean formula ρ on variables $x_1, \ldots, x_n$, the SAT problem consists in determining whether there is an *assignment* $\alpha \in \{0,1\}^n$ such that $\rho(x_1 := \alpha_1, \ldots, x_n := \alpha_n) = 1$.

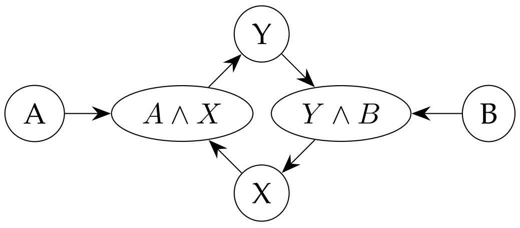


FIGURE 3.17 – Endless loop in dependencies for general boolean formulae.

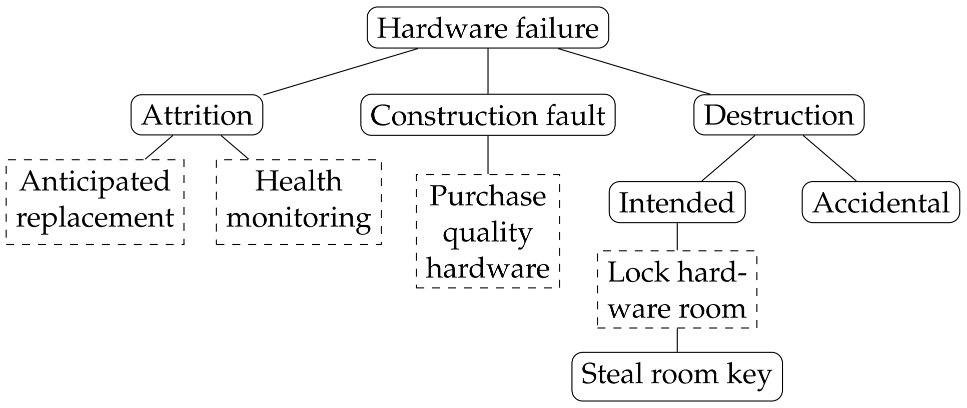


FIGURE 3.18 – A sample attack–defence tree. The dashed nodes correspond to defences that apply to the attack step above them.

be evaluated if $A \wedge X$ is known already. It is not so clear how to proceed in such a case: one solution is to set the likelihood to zero for all non-reachable nodes, because in fact the cycle can never be entered; however, this might not be sensible in all use-cases.

3.7.2 Attack defence trees

Attack trees were designed to understand an attack (or more generally, a risk scenario) and identify its origins, and thus constitute, to a certain extent, a special case of dependency graphs. While deterministic computations in the latter are hard and potentially take a lot of time (see Section 3.3), attack trees own much more structure and simplify the task by much. For instance, determining the probability that an attack succeeds is straightforward and can be achieved in an efficient way [125].

Since computations in attack trees are easy, more advanced concepts can be added, such as determining the optimal mediation for an attack, and its effectiveness. The generalised model is often referred to as attack–*defence* trees [126], since they do not only include the intermediate steps of the global attack, but also the respective defensive mechanisms. The latter model adopts the game-theoretic concept of two two players, opponent and proponent, who alternately try to defeat each other [127]. Figure 3.18 shows an example of an attack–defence tree.

Recent research work by Gadyatskaya et al. [49] shows how attack–defence trees can be combined with existent libraries (such as ISO/IEC 27002 [128]) to determine the security controls an organisation shall implement. Indeed, when a given set of controls is implemented, it will reduce the success probability of the attack, and thus the overall risk, but it also comes at a certain investment (namely implementation and maintenance costs of the respective solutions). The related optimisation problem consists in finding those controls that have the best return on investment.

In their paper ([49]), the authors semi-automatically embed the security controls from ISO/IEC 27002 [128] 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. Instead, in the following, a different solution is adopted.

Definitions

An attack–defence tree is defined [126] 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).

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.

Assumptions.

All attacks and defences in the tree are assumed to be independent. This assumption is generally made for attack (defence) trees, in order to simplify the computations. It might not reflect reality; dependency graphs have to be considered in the other case, instead.

Defences are allowed to protect from multiple attacks, possibly with different effectiveness values. In that case, however, they are implemented on a everywhere-or-nowhere basis, meaning that if it is implemented for one attack, it is automatically present for *all* applicable attacks.

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

$$p(\alpha) = (1 - e(\Delta)) \cdot \begin{cases} \prod_i p(i) & \text{if } \alpha \text{ is conjunctive} \\ 1 - \prod_i (1 - p(i)) & \text{if } \alpha \text{ is disjunctive.} \end{cases} \tag{3.5}$$

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

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

$$\text{ROSI} := \underbrace{\text{impact} \cdot (\text{initial probability} - \text{final probability})}_{\text{return (risk reduction)}} - \underbrace{\text{sum of costs,}}_{\text{investment}} \tag{3.6}$$

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.

For a set of defences Δ, the optimisation problem then reads as

$$\begin{array}{ll} \text{Find a strategy} & x : \Delta \to \{0,1\} \\ \text{that maximises} & \text{ROSI}(x), \end{array}$$

where $x(\delta) = 1$ denotes that defence δ should be implemented, and $x(\delta) = 0$ denotes that it should not.

Branch and bound algorithm

The optimisation problem can be solved in several ways. One possibility would be to turn ROSI(x) as defined in Equation (3.6) into a linear function and apply standard linear programming algorithms [129] on it. Such an approach has been proposed and described by Roy et al. [130]. While this technique works in theory, the size of the linear program exceeds the practical limits of feasibility very quickly.

The proposed algorithm is given in Algorithm 4 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 $x : \emptyset \to \{0,1\}$:

$$x_{opt} = \textsc{BnBa}\,(T, D, e, D_p, x)\,.$$

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 4 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.

Algorithm 4 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 $x_{\text{opt}}$ that maximises ROSI$(\cdot)$

if there is $\delta \in D_p$ that is no longer profitable (cf. Algorithm 5) then
    abort current recursion step
end if
if $D_p = D$ then
    $v \leftarrow$ ROSI$(x)$
    if $v$ is largest ROSI seen so far then
        $x_{\text{opt}} \leftarrow x$
    end if
else
    $\delta \leftarrow$ any defence not in $D_p$
    $D_p \leftarrow D_p \cup \{\delta\}$
    ′ Try selecting the defence
    $x(\delta) \leftarrow 1$
    BnBa$(T, D, e, D_p, x)$
    ′ Try not selecting the defence
    $x(\delta) \leftarrow 0$
    BnBa$(T, D, e, D_p, x)$
    ′ Remove $\delta$ again; this allows the re-use of $D_p$ among all recursive calls
    $D_p \leftarrow D_p \setminus \{\delta\}$
end if
```

Algorithm 5 Determine if a defence is profitable

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

1: if $x(\delta) = 0$ then
2:     return true
3: else
4:     ′ Extend $x$ to all of $D$
5:     $x(\delta') \leftarrow 0$ for all $\delta' \in D \setminus D_p$
6:     $x(\delta) \leftarrow 0$
7:     $v_0 \leftarrow \mathrm{ROSI}(x)$
8:     $x(\delta) \leftarrow 1$
9:     $v_1 \leftarrow \mathrm{ROSI}(x)$
10:    ′ $\delta$ is profitable iff the residual risk is lower when $\delta$ is implemented
11:    if $v_1 \cdot \mathcal{I} + c(\delta) < v_0 \cdot \mathcal{I}$ then
12:        return true
13:    else
14:        return false
15:    end if
16: end if
```

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 (3.5) 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.

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 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 4, 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 seconds in our implementation; so it would need $1.27 \cdot 10^{23}$ seconds ($4 \cdot 10^{15}$ years) to finish. On contrast, the optimised variant terminated within 895 seconds (15 minutes), having evaluated only $1,748,272$ combinations (which is approximately a 10^{-21} part).

Algorithm 4 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.

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.

3.8 Conclusion

This chapter introduced the dependency graph as a simple, visually speaking, and lightweight tool to model relationships between risk scenarios or assets. Since the graph is not assumed to be acyclic, the model can also be used in environments with interdependencies, such as in industrial control systems or critical infrastructures. In that sense, it is more general than competing models, and yet it can perfectly be used in parallel to most risk methodologies.

Apart from dependency graphs themselves, a major contribution of this chapter is Algorithm 1, which computes the risk described by such a graph in a provably efficient way. Indeed, as it turned out, any deterministic approach that we could think of is computationally too complex to serve as a basis for any usable algorithm, since their running time is exponential in the number of nodes and edges. Experiments reveal that this rapidly becomes a problem already for small graphs ($|V| \geq 30$).

In order to help risk assessors building such a dependency model, a taxonomy has been introduced to ease the elaboration of a graph, and to increase the consistence of the work. Moreover, a procedure is provided that describes how a dependency model can be automatically derived from an existing asset inventory. That way, the model cannot only adapt to changes of the threat landscape, but also incorporate modifications of the risk context in real time.

The model was developed with the intention of creating a tool that continuously computes and monitors the *current* risk faced by an organisation, taking all dependencies into account. The next chapter will be in charge for embedding the model into a risk monitoring framework, that allows it to be automatically updated when the risk situation changes in the field.

Chapter 4

Risk monitoring

4.1 Introduction

When it comes to securing a system or an entire organisation, it is crucial that security is a thought-out process, and does not only consist in deploying a tool or putting procedures into place. Instead, a good security strategy covers mechanisms that act in every stage of a hazard: namely *before* it arrives (preventive), *when* it occurs (detective, corrective), and *after* it has impacted the system (limitative).

In all cases, it is however not enough to only have such defences in place. Indeed, if they only work sporadically, or do not get triggered at all, the system is effectively vulnerable. Therefore, the most important thing for any successful security strategy is information; security responsibles should own, at all time, an exhaustive view of the security level in an organisation. Indeed, the earlier an incident is detected, the better it can get mitigated. And the longer a hazard or attack remains undetected, the longer an organisation is exposed to an increased risk.

Logging, monitoring solutions, and security information and event management (SIEM) systems thus constitute an integral part of a good security strategy. However, such sources of information are typically very technical and require decent knowledge of the underlying technology. Security managers, in contrast, adopt a high-level view and do not necessarily understand the extent of all possible error codes yielded by an appliance. This is even more applicable to industrial control systems, where status codes can be cryptic or undocumented.

4.1.1 Motivation

This chapter aims at creating the link between the high-level management view of risk, and the low-level and technical monitoring solutions. Since risk analyses already constitute a helpful tool for identifying major security issues in an organisation, the following sections will deliberate how this information can be turned into notions of risk, ready to be imported into the risk analysis. That way, the latter provides a holistic view for both organisational and technical issues.

Since logs are produced in real-time, the risk analysis essentially turns into a risk monitoring tool. That way, it does not only display the current risk

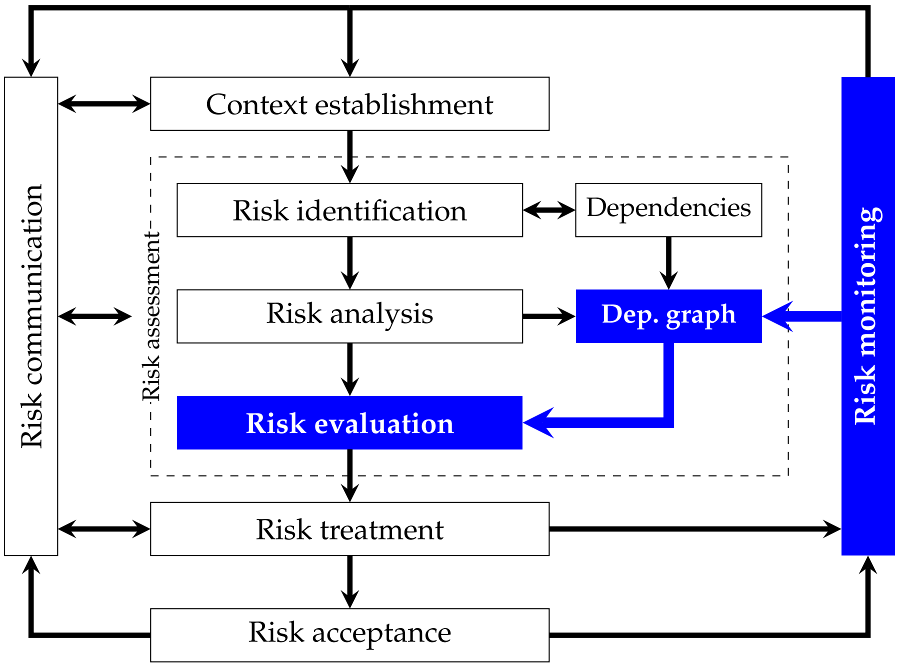


FIGURE 4.1 – Risk monitoring making use of the dependency model to automatically update the risk estimates in real-time.

at any moment, but it also regards technical issues in the context of the entire organisation. This will assist decision-makers in correctly assessing the criticality of an incident when it occurs. Moreover, such a 'dynamic' risk analysis allows risk assessors to simulate a risk scenario and identify the weakest part of their organisation.

4.1.2 Objective

The previous chapter has paved the way for a solid risk model, which additionally accounts for all dependencies at various levels. In the following, the very same model will be used to continuously update a risk analysis, as new real-time information is available about the monitored organisation. Figure 4.1 shows the subject of this chapter in the context of ISO 27005.

Whereas the model itself is generic enough to work with most risk methodologies, the risk monitoring platform will be tightly bound to a risk analysis tool. This is due to the fact that the dependency model needs to be able to interact with the latter, in order to update the risk analysis that is conducted with it. This thesis will therefore focus on a specific tool, namely TRICK Service[1]. It is to be noted, however, that the concepts developed in the following are not very restrictive and can also be applied to other risk management tools (possibly after source code modifications).

Particular care needs to be taken when importing logs, events, or alerts into a risk management platform. In fact, the necessary information usually originates from the field, a segregated network, an industrial environment,

[1] https://www.itrust.lu/trick-service/

or even from devices deployed outside the organisation's premises (which is the case for the smart grid, for instance). In such cases, any additional communication channel from and to the internal network constitutes a non-negligible risk of intrusion. Section 4.3 will present a non-intrusive strategy that does not expose either the internal network, nor the (possibly critical) field environment at a higher risk than usual.

Another major issue that needs to be addressed is the question how all kinds of status information (logs, errors, events, alerts, etc.) can be understood in *homogeneous* terms of risk. Indeed, some notifications will increase the risk level, while others already measure the risk level (e.g., likelihood of an intrusion) themselves. Section 4.2 will thus take care of providing a common denominator solution for the majority of cases.

4.1.3 Outline

This chapter is organised as follows.

- Section 4.2 addresses the fashion how risk is measured in the field. Most importantly, it describes the translation process of alerts and notifications into notions of risk.
- Section 4.3, in contrast, introduces the risk monitoring platform, and specifies how the 'measured risk' is reported from the field to the said platform.
- Section 4.4 then describes the link between the dynamically measured risk and the dependency model introduced in Chapter 3.
- Finally, the entire platform with all involved components is presented in Section 4.5.
- Section 4.6 deliberates further computations and simulations that can be made to obtain a deeper insight in the weakest links of the organisation.
- Ultimately, the chapter is concluded in Section 4.7.

4.2 Measure risk in the field

The intention of this section is to use the track records from existing security controls to infer the overall risk situation of the monitored system. These controls cover a complete spectrum of security appliances; examples include firewalls, load balancers, intrusion detection systems, spam filters, anti-virus suites, memory watchdogs, server uptime watchers, and many more. While some of these act as an active security guard, and others do not, all of them generate some kind of output which informs the responsible administrator about suspicious behaviour, which hints at a potentially increased risk exposure.

Risk, however, is a combination of multiple factors. As seen in Section 3.2.1, it is caused by the simultaneous presence of an external threat, a vulnerability, and an actual impact. In contrast, a monitoring tool can typically determine only one of these aspects. When reporting risk, one thus has to account for the situation where each of these is observed independently.

Moreover, another problem with the information reported by the security appliances resides in its heterogeneity. Not only do the several tools use a different format, but they also monitor different quantities: whereas some provide deterministic alerts which unambiguously state that a risk has occurred (such as the detection of an intrusion), others measure the riskiness of a situation (such as a high memory usage, or the availability of important updates).

On the one hand, due to that diversity, a dedicated utility needs to be developed for every security appliance, that reads the status information or log file, converts it into notions of risk, and reports it to the risk monitoring platform. On the other hand, for maximum flexibility, the latter platform should *not* depend on the security appliances in place. Therefore, it is unavoidable to define an intermediary notion of risk that covers all use-cases mentioned above.

4.2.1 Common-denominator risk

When risk is reported, several things need to be taken care of.

First, for security and confidentiality reasons, the risk reporting protocol should be unidirectional (also see Section 4.3 below). That is to ensure that the risk monitoring agent acts completely passive, and cannot compromise the possibly critical field network. This implies, however, that the risk monitoring platform cannot 'ask' the probes about their current status, but it needs to wait until the latter tell it.

Second, the reported risk should have an intrinsic notion of decay. Indeed, if an appliance observes suspicious behaviour, it will emit a corresponding notice – however, it will generally not tell if the situation is under control, again. There are a few exceptions to this, such as those that continuously measure risk (e.g., memory usage watchers), but for the majority of tools it should thus be possibly to specify a 'time of validity' for alerts when reporting risk.

Then, some alerts are more critical than others. Log files usually use the keywords 'info', 'warning', 'error', 'critical', etc. Available patches are often rated with the so-called Common Vulnerability Scoring System[2] (CVSS), a number between 0 and 10, with 10 being the most severe. Intrusion detection systems may yield a level of confidence. In all cases, more critical alerts should result in a higher risk.

Finally, as stated already, risk is defined as

$$\text{risk} = \text{threat} \times \text{vulnerability} \times \text{impact}.$$

[2]https://www.first.org/cvss/

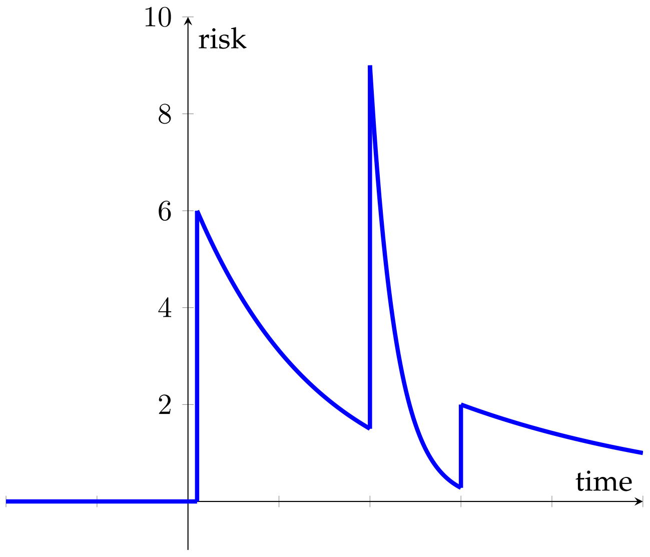


FIGURE 4.2 – Example evolution of a dynamically reported risk factor.

Since some tools measure the exposure to threats, while others rate the exploitability of the system itself, that difference should also be accounted for when risk is reported.

The proposed solution, which takes care of all of the criteria above, is the following. Each of the three risk factors (threat, vulnerability, impact) is modelled as a step-wise exponentially decaying function. It adopts its highest value when risk is reported, and decreases until a new value is reported, which then replaces the previous one. Figure 4.2 shows an example of such an evolution, where risk is reported three times (with initial values 6, 9, and 2, respectively).

Risk is then reported with the help of notifications, each of which appear as a peak in the risk evolution graph. More precisely, a notification is defined using the following parameters.

- The type of risk factor: threat, vulnerability, or impact. Each notification can describe only a single factor type. This choice does not cause any restrictions: when multiple types need to be reported, additional notifications can be sent.
- The severity $s \leq 0$, which expresses the initial level of risk that the system is exposed to. That value is understood in the context of threat, vulnerability, or impact, respectively. For threat-type notifications, s represents a likelihood and can be unbounded; for vulnerabilities, it represents the security of the system and must be ≤ 1 (see Section 3.2.1 on the components of risk).
- The half-life $h > 0$ states how fast the risk will decay, given that no new notification is reported. The half-life is defined as the time in seconds that needs to pass until the risk has decreased to exactly half of its initial value.

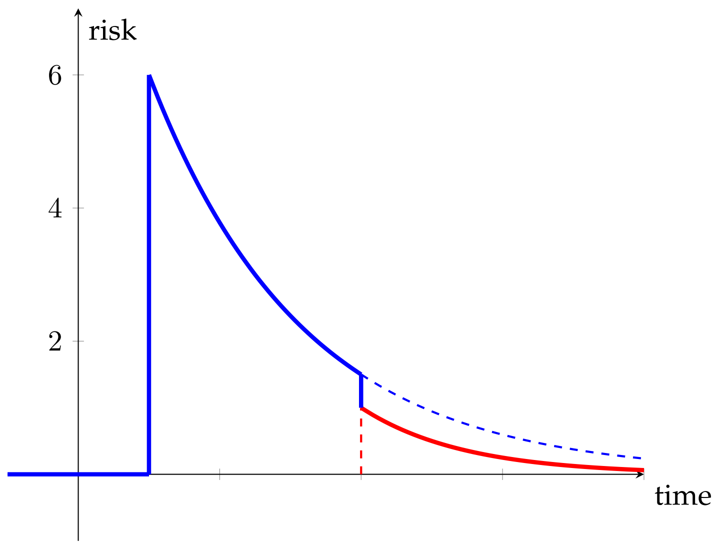


FIGURE 4.3 – A notification overridden by another one, due to the enforcement parameter being set (f=true).

For the parameters above, the risk factor will behave according to the function

$$t \mapsto s \cdot 2^{-(t-t_0)/h}$$

where t_0 denotes the time when the notification was reported.

4.2.2 Multiple alerts

When multiple alerts are reported one after the other for the same type, it is always the latest which dictates the behaviour; previous notifications are discarded.

However, in some circumstances, this approach is not optimal. Consider a monitoring utility, that observes a major breach and thus increases the threat level to some high value. A few moments afterwards, another smaller incident is detected, which consequently resets the threat level to a smaller value; the breach is then hidden by some unimportant event.

To prevent such misbehaviour from occurring, an additional parameter is added to the definition of notifications:

- The enforcement parameter $f \in \{\text{true}, \text{false}\}$ indicates whether the risk level described by the notification shall be enforced ($f = \text{true}$), or whether it should be discarded if its initial value is smaller than the residual risk from the previous notification ($f = \text{false}$).

The two possibilities are represented graphically in Figure 4.3 ($f = \text{true}$) and Figure 4.4 ($f = \text{false}$), respectively.

For simplicity reasons, it is always assumed that a notification dictates the 'new' risk level. Multiple notifications that progressively raise the overall risk level are not *directly* supported, for two reasons. First, the risk monitoring agent should be in control to specify the actual risk level (at least at

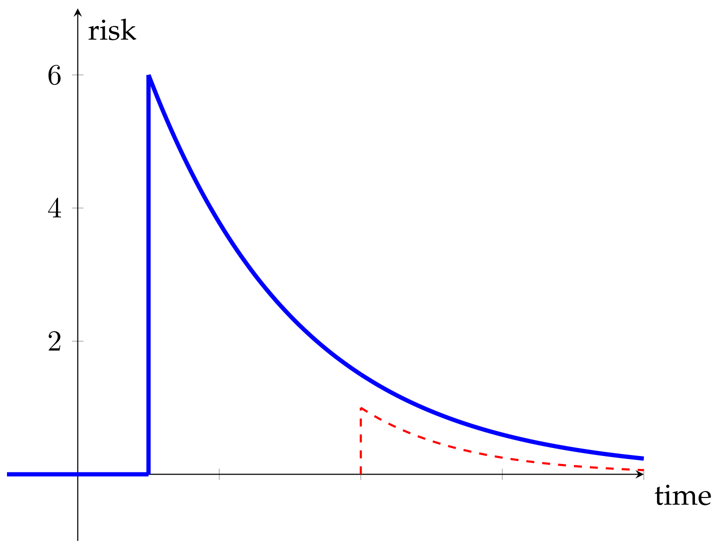


FIGURE 4.4 – A notification not overridden by another one, due to its smaller value and the enforcement parameter not being set (f=false).

the reporting times). It should not result from computations performed by risk monitoring platform, since the latter does not necessarily have all information available and might thus distort the view. Second, a monitoring tool might yield *a lot* of alerts for the same incident (firewalls are a good example, when they are under attack). In such a case, if each alert raises the risk level progressively, the resulting risk will be unbounded and thus meaningless.

It is to be noted that the risk level originating from *different* tools can still be combined. Section 4.4 will address this topic in greater detail.

4.2.3 Fixed risk level

Some appliances measure the risk level already on a regular basis, and do not need any decaying functionality. For instance, when evaluating the criticality of available patches, the risk level should stay high as long as the patch is not applied.

This case is really a special case of the model introduced above, though. In fact, by choosing an infinity half-life $h := \infty$, the decaying function looks like

$$t \mapsto s \cdot 2^{-(t-t_0)/\infty} = s \cdot 2^0 = s.$$

In words, infinite (or very large) half-lives exactly represent the case where the risk level should stay at its position.

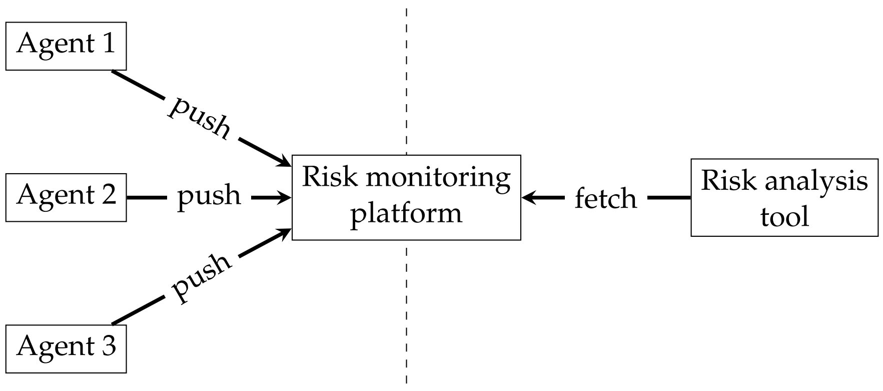


FIGURE 4.5 – An intermediary risk monitoring platform for storing risk indicator values, separating agents from the risk analysis.

4.3 Risk reporting

Rendering a risk analysis dynamic is not so easy than it seems. A naive strategy would consist in the risk monitoring agents updating those parts of the analysis which concern them. However, such an approach requires the agents to know the structure of the risk analysis already in advance. If a risk assessor decides at a later point to reorganise the analysis, all agents potentially need to be reconfigured. However, monitoring tools usually reside in critical networks; when they can be configured (so modified) from the outside, the respective network is exposed to an unnecessary high risk.

Therefore, risk information collected by agents cannot be directly sent to the risk analysis in question. Instead, an intermediary risk monitoring platform is needed, which merely serves as a storage space for the recorded indicator values. It is depicted in Figure 4.5. The platform acts purely passively and thus constitutes a de-militarised zone: on the one hand, agents push the latest risk information, and on the other hand, the risk analysis tool fetches the values that it needs. That way, no direct communication is possible from and to the agents (and thus the critical networks).

The principle is the following:

1. A monitoring tool records and estimates risk as in Section 4.2.
2. That risk information is reported to the risk monitoring platform, and stored into a named parameter. The name is picked by the monitoring agent and can be arbitrary, but should be used consistently. It does not need to be unique; in contrast, different tools should even use the same name if a similar quantity is reported, to enable risk aggregation afterwards (see Section 4.3.4).
3. When risk is reported multiple times for the same named parameter by the same agent, its value will be overridden according to Section 4.2.2. Parameters reported by *other* agents are not affected by this rule.

4. Finally, risk assessors link the risk analysis tool with the monitoring platform, so that the risk analysis uses these named parameters instead of pre-filled risk information. This is further elaborated in Section 4.4.

4.3.1 Protocol

The risk monitoring platform exposes an HTTP API[3] which the agents can connect to and report risk information to.

HTTP APIs have the advantage that they are widely supported by many programming languages and tools, without requiring complicated or proprietary libraries to be installed. Moreover, system administrators are rather willing to open firewall ports for HTTP traffic which they understand, than for cryptic protocols for which they cannot see what it transmitted.

The API defines a single method `POST /notify` that can be invoked with the following parameters. When calling this method with risk information as defined in Section 4.2, the latter will be stored as a named parameter.

Parameter	**Value**	**Description**
`sender_id`	string	A unique identifier for the monitoring agent.
`time`	number	The UNIX timestamp when the risk was reported. Typically corresponds to `now()`.
`param`	string	The name of the parameter that will hold the risk information.
`severity`	number	The s parameter from Section 4.2.
`half_life`	number	The h parameter from Section 4.2.
`force`	boolean	The f parameter from Section 4.2.

4.3.2 Authentication

To prevent spoofing attempts, it is recommended to require authentication from monitoring tools when they try to use a specified `sender_id`. The risk monitoring platform can generate an API access token for each agent that is expected to report risk, and bind it to a single `sender_id`. That way, it is guaranteed that only verified tools can interfere with the risk management process.

On the protocol-side, authentication can be achieved using the HTTP header '`Authorization: Basic`'.

4.3.3 Interaction with risk analysis

At the other side, the risk analysis tool retrieves those named parameters in order to use their value in a risk analysis. That is, instead of manually

[3]Application Programming Interface, a set of defined methods for interacting with a component.

estimating the likelihood of a risk scenario, a risk assessor may link it with a named parameter, so that the likelihood reported by a risk monitoring agent is applied, instead.

Note that the low-level agents do not know anything about the risk management process. In particular, they do not have a notion of risk scenarios and assets, and thus are not automatically linked to the appropriate place in the risk analysis. It is thus the risk assessor's task to associate the named parameters with the corresponding assets and scenarios.

In fact, when risk is reported, it induces a value function v for each monitoring agent $a \in \mathcal{A}$ (identified by its `sender_id`) and each parameter $p \in \mathcal{P}$,

$$\begin{aligned} v : \mathcal{A} \times \mathcal{P} \times \mathbb{R}_+ &\to \mathbb{R} \\ (a, p, t) &\mapsto v_{a,p}(t), \end{aligned}$$

which associates the current risk value to every time t for a given agent and parameter (a, p). The function $t \mapsto v_{a,p}(t)$ has the piece-wise exponentially decaying shape described in Section 3.2 above.

Risk assessors then use the expression $v_{a,p}(t)$ (for the appropriate parameter (a, p)) in the risk analysis tool, which automatically retrieves the current value (i.e., for $t :=$ `now()`) and plugs it into the risk analysis.

Mathematically speaking, the map $t \mapsto v_{s,n}(t)$ constitutes a time series. As will be discussed in Section 4.6, further properties can be determined from historical values (such as an evolution of risk).

4.3.4 Aggregating risk levels

As argued before, the low-level monitoring agents are not supposed to coordinate with one another for security (or technical) reasons. It may be of interest, however, to aggregate the risk information from several sources. For instance, if multiple intrusion detection system agents report intrusion alerts, one would prefer to include all of them in the same spot in the risk analysis.

The solution consists in allowing users of the risk analysis tool to not only specify a named parameter s, n, but to also let them specify an *aggregation* operation (such as the minimum, maximum, sum, or average of given named parameters).

Formally, an aggregation operation is defined as a 3-tuple $(\bar{a}, p, o)$ consisting of an identifier prefix $\bar{a}$, a parameter p, and a function $o : \mathbb{R} \times \cdots \times \mathbb{R} \to \mathbb{R}$. Examples for o are the sum, minimum or maximum of a sequence of numbers. It proceeds as follows:

1. The set $\alpha = \{a \in \mathcal{A} \mid a$ **startswith** $\bar{a}\}$ is determined.
2. For each $a \in \alpha$ and $p \in \mathcal{P}$, the current numeric value of $v_{a,p}(t_{\text{now}})$ is determined.
3. All these numeric values are aggregated using the function $o(\cdot)$.

Intuitively, an aggregation combines risk information from all agents whose identifier starts with the same given string. For that reason, the identifiers of agents should be meaningfully structured. For example, multiple instances of a distributed network intrusion detection system could be named

- `datacenter/network/ids/1`
- `datacenter/network/ids/2`
- `datacenter/network/ids/3`

and so on. In that case, all intrusion alerts could be aggregated, for instance, using the aggregation operation

(`datacenter/network/ids/`, `intrusion`, **max**),

which would yield the maximum threat level for the '`intrusion`' parameter reported by all IDS agents.

4.4 Dynamic risk computation

Once the risk values are measured in the field and reported to the risk monitoring platform, they can be fetched by the risk analysis tool and used in risk assessments. This section shows how the dependency model can be rendered dynamic in the sense that it includes the risk indicators that are being reported in real time.

4.4.1 Risk in matrix form

In Section 3.2.4, the risk of a scenario σ was determined to be

$$R(\sigma) := \sum_{\alpha \in \text{ante}(\sigma)} \left(\sum_{v \in \text{rt}(V_{\mathcal{G}})} \mathbb{P}[v \rightsquigarrow \alpha] \cdot \mathbb{L}(v) \right) * \mathbb{I}(\alpha). \tag{4.1}$$

This formula involves the two maps $\mathbb{L}$ and $\mathbb{I}$, referring to the likelihood and impact of security events encoded in the graph, respectively. Moreover, it contains the causing probabilities $\mathbb{P}[v \rightsquigarrow \alpha]$, denoting the probability that an event v eventually causes another event α.

However, the expression above can also be written out in matrix form. In fact, the involved functions

$$\begin{aligned} \mathbb{P} &: \text{rt}(V_{\mathcal{G}}) \times V_{\mathcal{G}} \to [0,1] \subset \mathbb{R} \\ \mathbb{L} &: \text{rt}(V_{\mathcal{G}}) \to \mathbb{R} \\ \mathbb{I} &: V_{\mathcal{G}} \to I \end{aligned}$$

can also be viewed as matrices and vectors

$$\begin{aligned}\mathbb{P} &\in \mathbb{R}^{\mathrm{rt}(V_\mathcal{G})\times V_\mathcal{G}}\\ \mathbb{L} &\in \mathbb{R}^{\mathrm{rt}(V_\mathcal{G})\times 1}\\ \mathbb{I} &\in I^{V_\mathcal{G}\times 1}.\end{aligned}$$

Equation 4.1 then reads as

$$\begin{aligned}R(\sigma) :=& \sum_{\alpha\in\mathrm{ante}(\sigma)} \left(\mathbb{L}^\top \cdot \mathbb{P}\right)(\alpha) * \mathbb{I}(\alpha)\\ =& \left(\mathbb{L}^\top \cdot \mathbb{P}\right)\Big|_{\mathrm{ante}(\sigma)} * \mathbb{I}\big|_{\mathrm{ante}(\sigma)},\end{aligned} \tag{4.2}$$

where the vertical bar $\cdot\big|_{\mathrm{ante}(\sigma)}$ denotes the restriction of the $1\times V_\mathcal{G}$ and $V_\mathcal{G}\times 1$ matrices to the respective domains $1\times \mathrm{ante}(\sigma)$ and $\mathrm{ante}(\sigma)\times 1$.

$R(\sigma)$ denotes the risk resulting from a *single* scenario σ; similarly, the total risk for the *entire* organisation can be obtained by

$$R := \sum_{\alpha\in V_\mathcal{G}} \left(\sum_{v\in\mathrm{rt}(V_\mathcal{G})} \mathbb{P}[v \rightsquigarrow \alpha]\cdot \mathbb{L}(v)\right) * \mathbb{I}(\alpha),$$

(note the sum over *all* vertices $\alpha \in V_\mathcal{G}$). In matrix form:

$$R = \left(\mathbb{L}^\top \cdot \mathbb{P}\right) * \mathbb{I}.$$

4.4.2 Dynamic risk

In Equation 4.2 for the total risk of a scenario, all involved quantities can be made dynamic. If fact, each of them matches one of the factors that risk is composed of – threat, vulnerability, and impact.

- The likelihoods $\mathbb{L}$ correspond to the risk information for threats reported by risk monitoring agents,
- while the vulnerabilities reflect in increased edge probabilities $p(\cdot)$ of the dependency graph. Indeed, if a component is exploitable, then there is an increased probability that the associated security event is caused. Ultimately, increased edge weights p also result in higher probabilities $\mathbb{P}$, of course.
- Finally, the impacts $\mathbb{I}$ correspond exactly to the consequences accompanying a risk scenario.

When risk information is reported, it is thus enough to update the corresponding value in either $\mathbb{L}$, $p(\cdot)$, or $\mathbb{I}$ (whichever applies).

Risk assessors thus need to associate the $\mathbb{L}$, p, $\mathbb{I}$ values in the risk model to the respective dynamic parameters. This is achieved by typing the parameter name as a place holder (e.g. "`intrusion`") instead of a number (e.g. $0.1/y$), such that the appropriate value is retrieved from the risk monitoring platform. For more information, see Section 4.3 that precedes.

4.4.3 Algorithm

Algorithm 6 outlines the procedure for computing the dynamic risk in real-time; its running time is analysed below.

Intuitively, the algorithm first updates $\mathbb{L}$, $\mathbb{I}$, and $p(\cdot)$ by retrieving the latest values from the risk monitoring platform according to Section 4.2. For performance reasons, it will internally operate in a slightly different way. A better approach consists in maintaining a cached version of the above quantities, and only retrieve the latest notifications instead – if no new notifications are available since the last execution of the algorithm, it will not update anything.

After it is up to date with what has been reported, the algorithm proceeds with inferring the *eventually causing probabilities* $\mathbb{P}$ from the probability map $p(\cdot)$. Recall that $p(\alpha, \beta)$ denotes the probability that α causes β directly, whereas $\mathbb{P}[\alpha \rightsquigarrow \beta]$ represents the probability that there is chain from α to β.

Finally, the algorithm re-computes the resulting risk according to Equation 4.2 above.

Algorithm 6 Compute dynamic risk.

```
1: for $v \in \text{rt}(V_{\mathcal{G}})$ do
2:     if $\mathbb{L}(v)$ is a named parameter then
3:         retrieve the latest notification for $\mathbb{L}(v)$
4:         calculate $\mathbb{L}(v)$ for $t = t_{\text{now}}$
5:     end if
6: end for

7: for $v \in V_{\mathcal{G}}$ do
8:     if $\mathbb{I}(v)$ is a named parameter then
9:         retrieve the latest notification for $\mathbb{I}(v)$
10:        calculate $\mathbb{I}(v)$ for $t = t_{\text{now}}$
11:    end if
12: end for

13: for $\alpha \in \text{rt}(V_{\mathcal{G}}), \beta \in V_{\mathcal{G}}$ do
14:     if $p(\alpha, \beta)$ is a named parameter then
15:         retrieve the latest notification for $p(\alpha, \beta)$
16:         calculate $p(\alpha, \beta)$ for $t = t_{\text{now}}$
17:     end if
18: end for

19: Run Algorithm 1 with the new probability map $p(\cdot)$ to obtain $\mathbb{P}$

20: for each risk scenario $\sigma \in V_{\mathcal{G}}$ do
21:     Compute risk as $\text{risk}(\sigma) = \left(\mathbb{L}^\top \cdot \mathbb{P}\right)\big|_{\text{ante}(\sigma)} * \mathbb{I}\big|_{\text{ante}(\sigma)}$
22: end for
```

4.4.4 Running time

To analyse the running time, it is assumed that the retrieved notifications are appropriately cached in the implementation of Algorithm 6. That way, unnecessary interactions between the risk analysis tool and the risk monitoring platform are avoided.

Proposition 4.1. *Suppose the dependency graph consists of n nodes and m edges. Then the running time of Algorithm 6 is*

$$\mathcal{O}\left(n^3 + n \cdot m \cdot \ln\left(\frac{2n}{\delta}\right) \cdot \varepsilon^{-3}\right),$$

where ε and δ denote the relative error and the error probability from Algorithm 1, respectively.

Proof. Suppose there are N pending notifications. Lines 1–6 in Algorithm 6 require at most $\max(N, n)$ iterations, since the loop is executed at most once per node, and at most once for every notification (if the caching solution as described above is implemented). Moreover, each iteration requires constant running time, since calculating $\mathbb{L}(v)$ follows a deterministic formula that can be imminently computed.

The same is true for lines 7–12 and 13–18, respectively, except that for the latter, there are at most $\max\left(N, n^2\right)$ iterations.

The running time of Algorithm 1 is given by Proposition A.2, namely

$$\mathcal{O}\left(n \cdot m \cdot \ln\left(\frac{2n}{\delta}\right) \cdot \varepsilon^{-3}\right).$$

Finally, the risk is computed for each of the at most n risk scenarios. Since the computation of risk consists of matrix multiplications, their running times depend on the matrix dimensions. From Equation 4.2, one can see that two sums hide behind the matrix product, so the running time is $O\left(n^2\right)$. The overall time for lines 20–22 is thus $O\left(n^3\right)$.

The statement of the proposition follows from $\max\left(N, n^2\right) \in O\left(n^3\right)$. □

In the few use-cases that were considered in the course of the thesis, dependency graphs typically have 20–100 nodes and 50–300 edges. The error parameters are typically chosen as $\varepsilon = 0.1$ and $\delta = 0.01$. For these values, $n \cdot m \cdot \ln\left(\frac{2n}{\delta}\right) \cdot \varepsilon^{-3} \gg n^3$, so the overall running time simplifies to

$$\mathcal{O}\left(n \cdot m \cdot \ln\left(\frac{2n}{\delta}\right) \cdot \varepsilon^{-3}\right).$$

For larger graphs ($n = 100$, $m = 300$) such simulations take around 30 seconds; for smaller graphs ($n = 20$, $m = 50$) the execution time is negligible (less than 1 second).

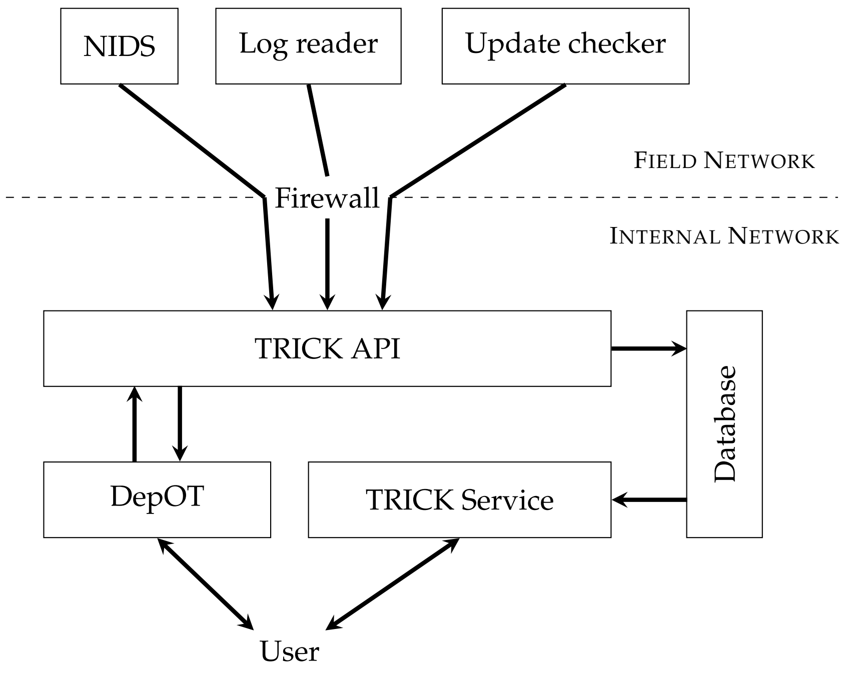


FIGURE 4.6 – Interaction of risk monitoring components.

4.5 Risk monitoring platform

The overall risk monitoring platform thus consists of agents that report real-time risk information to a central storage place, which can be accessed by the risk analysis tool to keep all analyses up to date. The entire platform is depicted in Figure 4.6.

4.5.1 Agents

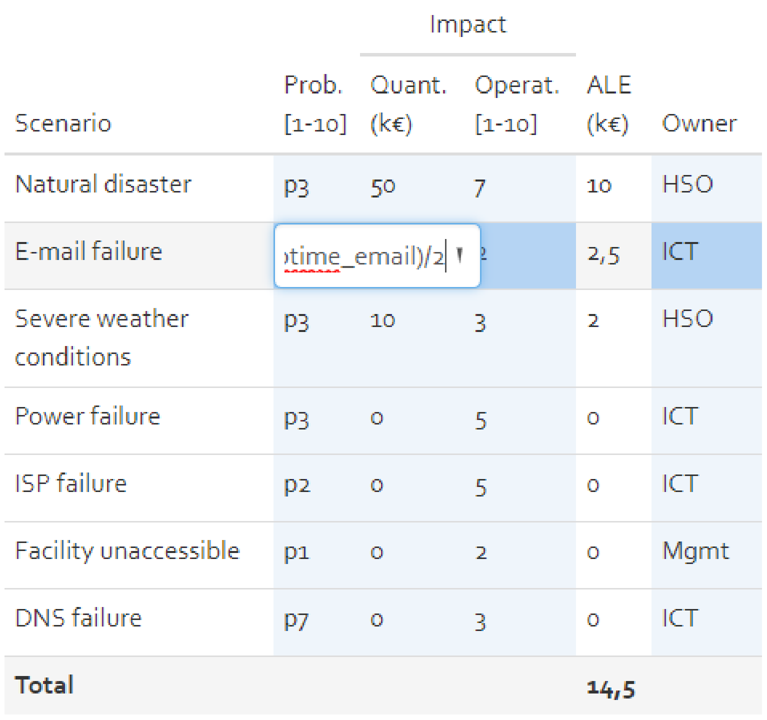
Risk monitoring agents measure the current risk according to Section 4.2. They typically reside in other networks, and need to pass a firewall to connect to the risk monitoring platform. The said firewall needs to be configured to let those flows pass. However, since the communication is unidirectional, there is only a small additional risk that exposes the system to new threats.

More words on concrete implementations of such agents will be spent in Chapter 5. The most prominent use-case will be a network intrusion detection system for industrial systems, that finds a particular family of advanced persistent threats (APT).

4.5.2 TRICK Service

The web-based tool 'TRICK Service'[4] is a full-featured risk management tool that assists risk assessors throughout the entire risk assessment process.

[4] https://www.itrust.lu/trick-service/



		Impact			
Scenario	Prob. [1-10]	Quant. (k€)	Operat. [1-10]	ALE (k€)	Owner
Natural disaster	p3	50	7	10	HSO
E-mail failure	›time_email)/2		2	2,5	ICT
Severe weather conditions	p3	10	3	2	HSO
Power failure	p3	0	5	0	ICT
ISP failure	p2	0	5	0	ICT
Facility unaccessible	p1	0	2	0	Mgmt
DNS failure	p7	0	3	0	ICT
Total				**14,5**	

FIGURE 4.7 – Linking a named parameter to a likelihood in TRICK Service with the use of formulae.

It maintains an asset inventory together with a set of risk scenarios, and lets users estimate the risk associated to each of them (see Figure 4.7).

In the context of this doctoral thesis, the risk analysis tool was extended to also support dynamic risk. Indeed, the named parameters that are created by the risk monitoring platform (see Sections 4.2 and 4.3) have been made available in the tool. The principle is similar to the one known from spreadsheet applications: instead of specifying a concrete (risk) value, users can fill in an entire formula into the respective fields. For instance, Figure 4.7 shows a likelihood that has been bound to a named parameter through the use of such a formula. In the example, the complete expression is

```
(max(internal_email_) + external_uptime_email) / 2
```

representing the average of the risk measured internally (by any agent on the e-mail server) and the up-time of the server (determined by an external service provider).

More complicated expressions are possible, that make use of:

- Common mathematical operations ($+$, $-$, $\star$, $/$, ^).
- Parentheses for grouping operations (e.g., $5 \cdot (1+2)$).
- Named parameters that have been reported to the risk monitoring platform.
- Pre-defined functions (such as `min`, `max`, `avg`) that aggregate risk from several sources, as described in Section 4.3.4.

Algorithm 7 documents (in pseudo code) how such an expression can be parsed. The keyword '`resolve`' is used to retrieve a named parameter (or a set thereof, in the case of aggregation) and to resolve it to the corresponding value, according to the exponentially decaying model introduced in Section 4.2.

4.5.3 TRICK API

The core part of the risk monitoring platform consists of the HTTP/JSON API[5] that receives risk information from the various sources. The procedure thereof is described in Section 4.3. It mainly acts as a central storage place for all notifications reported by the monitoring agents, and provides this information to the risk analysis tool.

In the context of this thesis, risk monitoring is implemented as an add-on for the risk analysis tool, since the latter already happened to feature an API before dynamic risk was added. Therefore, the exchange between the risk monitoring platform ('TRICK API') and the risk analysis tool ('TRICK Service') is entirely done with the help of a shared database; since both tools have (read/write) access to the same data, no further interaction is necessary.

4.5.4 DepOT

The TRICK Service API features additional methods for retrieving and updating information about a risk analysis. External tools may then fetch the asset inventory linked to an analysis that is stored in TRICK Service. For instance, dependency modelling tools (such as DepOT, presented in Section 3.5.2) can integrate with the risk analysis tool and maintain a structured representation of the inventory in parallel to the latter.

Since TRICK Service on its own does not support dependencies, computing the dependency-aware risk (see Equation 4.2 in Section 4.4) as

$$\left(\mathbb{L}^\top \cdot \mathbb{P}\right)\Big|_{\text{ante}(\sigma)} * \mathbb{I}\big|_{\text{ante}(\sigma)}$$

is not viable without significantly modifying the software. Instead, a lighter approach was adopted.

Since TRICK Service was extended to support the use of formulae, DepOT expresses the modelled dependencies $\mathbb{L}^\top \cdot \mathbb{P}$ explicitly as functions f_i with

$$\begin{aligned}\mathbb{L}[v_1] &= f_1(\mathbb{L}[v_2], \mathbb{L}[v_3], \ldots, \mathbb{L}[v_n])\\ \mathbb{L}[v_2] &= f_2(\mathbb{L}[v_1], \mathbb{L}[v_3], \ldots, \mathbb{L}[v_n])\\ &\ldots\end{aligned}$$

without computing their actual value. Since the matrix product is a linear operation, the expressions f_i above only involve casual additions and multiplications. The thus obtained formulae are then imported into TRICK Service via its API.

[5]An application programming interface (API) that accepts data in JSON format over an HTTP communication channel.

Algorithm 7 Algorithm for parsing an expression that resolves named parameters to the represented dynamic risk.

```
func parse():
        return parse_sum()

func parse_sum():
        val = parse_product()
        while next token is "+" or "-":
                op = read token    # either + or -
                val = val 'op' parse_product()
        return val

func parse_product():
        val = parse_power()
        while next token is "*" or "/":
                op = read token    # either * or /
                val = val 'op' parse_power()
        return val

func parse_power():
        val = parse_parentheses()
        if next token is "^":
                read token    # "^"
                val = val ^ parse_parentheses()
        return val

func parse_parentheses():
        if next token is "(":
                read token    # "("
                val = parse_sum()
                read token    # ")"
        else:
                val = parse_literal()
        return val

func parse_literal():
        tok = read token
        # Token can be a function name,
        # number, or named parameter
        if next token is "(":
                read token    # "("
                prefix = read token
                vals = resolve 'prefix'
                val = call tok ( vals )
                read token    # ")"
        else if tok is number:
                val = tok
        else: # otherwise it is a parameter
                val = resolve 'tok'
        return val
```

Note how dependencies integrate nicely with the concept of dynamic risk analyses. As a consequence, they also get automatically updated when new risk information is reported.

4.6 Additional computations

4.6.1 Determine the most critical risk

In a risk analysis, one is typically interested in the total risk resulting from a scenario. This has been thoroughly discussed in the previous sections, notably in Section 3.3.

That way, one can easily determine the most critical risk scenarios. It is enough to compute the total risk for each scenario σ as given by Equation 3.2, namely

$$\sum_{\alpha \in \mathrm{ante}(\sigma)} \left(\sum_{v \in \mathrm{rt}(V_{\mathcal{G}})} \mathbb{P}[v \rightsquigarrow \alpha] \cdot \mathbb{L}(v) \right) * \mathbb{I}(\alpha),$$

and sort the risk scenarios by decreasing risk.

Note that the latter equation only reveals the most severe *symptoms*. Notwithstanding, it is evenly as important to understand where the problems come from, especially when it comes to risk treatment. Fortunately, the probability matrix $\mathbb{P}$ holds much more information than just the total risk. For instance, in order to determine the most likely origin of security incidents for a given scenario $\sigma \in V_{\mathcal{G}}$, it is enough to consider

$$\operatorname*{arg\,max}_{c \in \mathrm{rt}(V_{\mathcal{G}})} \mathbb{P}[c \rightsquigarrow \sigma],$$

so the root cause c which eventually leads to σ with the highest probability.

Similarly, one can also determine the criticality of a root cause in terms of risk. In contrast to the case for risk scenarios, one is not interested in the effects of a particular incident, but rather in all possible incidents (and their consequences) that can arise from a specific root cause. Very much in the spirit of Equation 3.2, the risk associated to a root cause $c \in \mathrm{rt}(V_{\mathcal{G}})$ can be determined as

$$\sum_{\alpha \in \mathrm{desc}(c)} (\mathbb{P}[c \rightsquigarrow \alpha] \cdot \mathbb{L}(c)) * \mathbb{I}(\alpha),$$

where $\mathrm{desc}(c)$ denotes the set of all nodes that can be eventually caused by c (or in graph language, the set of descendants from c), including c itself.

4.6.2 Determine the most likely cascade effect

Even though cascade effects are a major concern, especially in dependency-rich environments such as industrial control systems, it is not trivial to foresee cascade effects at all, let alone identify the most critical one.

When dependencies are correctly modelled, part of these questions can be answered using the dependency graph $\mathcal{G} = (V_\mathcal{G}, E_\mathcal{G}, p)$. Indeed, a cascade effect is a chain of causal relationships, which correspond to paths of edges in the graph. The criticality of a chain then correlates with the accumulated probability of the related path $(v_0, v_1, \ldots, v_n)$, and is formally defined as

$$\prod_{i=1}^{n} p\left(v_{i-1}, v_i\right).$$

One may be either interested in the most critical cascade effect that *starts* at a given root cause, or in the one that *leads* to a given risk scenario. Either way, the problem consists in finding the path that has the largest probability among all paths starting at (or ending in) a certain node.

To solve the problem, it is reduced to a 'shortest path' problem, for which well-known solutions exist. First, define an alternative weighting map

$$p'(\alpha, \beta) := -\log\left(p(\alpha, \beta)\right) \geq 0.$$

for all nodes $\alpha, \beta \in V_\mathcal{G}$. Then the probability of a path in $(V_\mathcal{G}, E_\mathcal{G}, p)$ corresponds exactly to the total weighting of the same path in $(V_\mathcal{G}, E_\mathcal{G}, p')$, up to the homomorphism $x \mapsto -\log(x)$. Indeed,

$$\underbrace{-\log\left(\prod_{i=1}^{n} p\left(v_{i-1}, v_i\right)\right)}_{\text{path probability}} = \underbrace{\sum_{i=1}^{n} p'\left(v_{i-1}, v_i\right)}_{\text{path weighting}}.$$

Since $x \mapsto -\log(x)$ is order-reversing, a *maximum* probability path in the original graph $(V_\mathcal{G}, E_\mathcal{G}, p)$ corresponds to a *minimum* weight path in the artificial graph $(V_\mathcal{G}, E_\mathcal{G}, p')$.

The problem can now be solved with any algorithm for determining the shortest path in a graph, including Dijkstra [131], Fredman and Tarjan [132], or Floyd–Warshall [133].

Algorithm 8 is based on Floyd–Warshall [133] and finds the most likely cascade effect for a given root cause c.

4.6.3 Evolution of risk

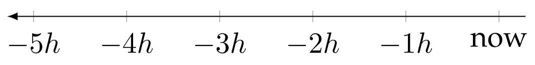
With dynamic risk management, the risk analysis can get updated at any time of the day. So it might be the case that incidents are reported when the tool is not surveilled, for instance at night or over the week-end. Displaying the *historical* risk thus helps security managers to assess the criticality of past incidents, as well.

A time line is the simplest way of representing the evolution of risk over time. However, incidents from a few months ago might not be as important as very recent ones, even though they might still be relevant (e.g., in order to compare a current incident with a past major breach). Since the screen size only allows displaying a limited time period in a graph, a linear time

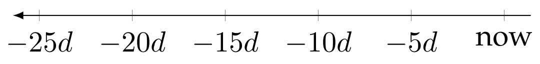
Algorithm 8 Algorithm for finding the most critical cascade effect.

Input: Dependency graph $G = (V, E, p)$
Input: A root cause $c \in V$
Output: A most probable path $(c, v_1, \ldots, v_m)$ starting at c

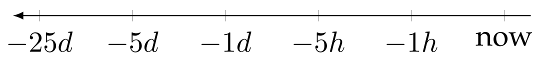
```
1: ′ Floyd–Warshall algorithm
2: Set $d(i, j) := \infty$    for all $1 \le i, j \le |V|$
3: Set $\xi(i, j) :=$ nil    for all $1 \le i, j \le |V|$
4: for $(v_i, v_j) \in E$ do
5:     $d(i, j) := -\log p(v_i, v_j)$
6:     $\xi(i, j) := j$
7: end for
8: for $k = 1$ to $|V|$ do
9:     for $i = 1$ to $|V|$ do
10:        for $j = 1$ to $|V|$ do
11:            if $d(i, j) > d(i, k) + d(k, j)$ then
12:                $d(i, j) := d(i, k) + d(k, j)$
13:                $\xi(i, j) := \xi(i, k)$
14:            end if
15:        end for
16:    end for
17: end for

18: ′ Determine most likely consequence
19: $\sigma := \arg\max_{v \in V} d(c, v)$

20: ′ Determine shortest path from $c$ to $\sigma$
21: Initialise list $P := (c)$
22: $x := c$
23: while $x \ne \sigma$ do
24:     $x := \xi(x, \sigma)$
25:     $P := P \cup (x)$
26: end while
27: return $P$
```



(A) Linear time line with high level of precision (minutes) and small time coverage (5 hours).



(B) Linear time line with low level of precision (days) and big time coverage (1 month).



(C) Logarithmic time line with both high level of precision (minutes) for the recent past, and big time coverage (1 month).

FIGURE 4.8 – Comparison of linear and logarithmic time lines.

line can only show events up to a certain point in the past. *Logarithmic* plots, in contrast, can visualise the current situation with an appropriate level of details in combination to the very distant past. The three possibilities are depicted for comparison in Figure 4.8. For the reasons mentioned, option (C) is the most appropriate one.

Since putting all historical data in a single graph would be infeasible from a computational point of view, averaging techniques are applied beforehand to reduce the amount of information to display.

Algorithm 9 Algorithm for computing the average risk in a logarithmic time line.

```
Input: Time points $t_0 < t_1 < \cdots < t_n$
Input: List of risk notifications $\mathcal{N}$ sorted by reporting time
Output: Average risk $R_i$ on each interval $[t_{i-1}, t_i]$
 1: for $i = 1$ to $n$ do
 2:     $R_i := 0$
 3:     for $j = 1$ to $|\mathcal{N}|$ do
 4:         ′ $T_j$ denotes the time when $(\mathcal{N})_j$ was reported
 5:         ′ $H_j$ denotes the half-life time of $(\mathcal{N})_j$
 6:         ′ $S_j$ denotes the severity of $(\mathcal{N})_j$
 7:         if $T_j < t_i \wedge T_{j+1} > t_{i-1}$ then
 8:             $A := \max\{t_{i-1}, T_j\}$    ′ Left boundary
 9:             $B := \min\{t_i, T_{j+1}\}$    ′ Right boundary
10:             $R_i := R_i - \frac{1}{t_i - t_{i-1}} \cdot S_j \cdot \frac{H_j}{\ln 2} \cdot \left(2^{-\frac{A-T_j}{H_j}} - 2^{-\frac{B-T_j}{H_j}}\right)$
11:         end if
12:     end for
13: end for
```

Algorithm 9 shows how the average risk can be computed for an arbitrary time line. It proceeds by computing the integral of the risk induced by each notification over time. According to Section 4.2, given such a notification,

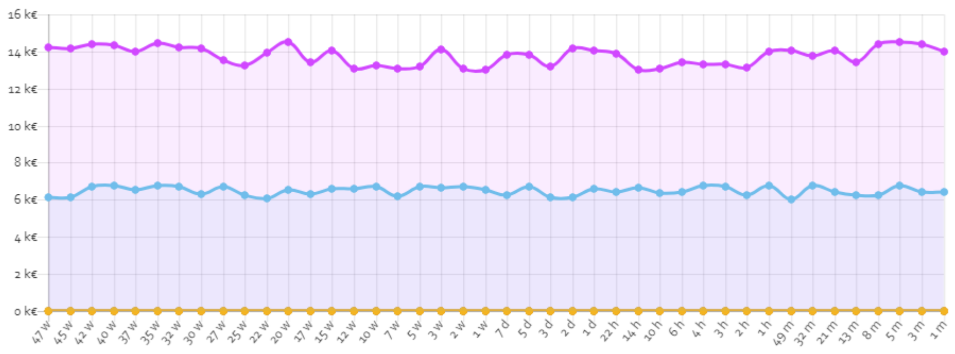


FIGURE 4.9 – Evolution of (quantitative) risk visualised in a logarithmic time line in TRICK Service.

its risk is given by

$$t \mapsto S \cdot 2^{-(t-T)/H}$$

where T is the time when it was reported, H is its half-life time, and S its severity. Since consecutive notifications override each other, the j-th notification does not apply all the time, but only between time T_j and T_{j+1}. Therefore, the total risk contributed by the j-th notification is given by

$$\int_{T_j}^{T_{j+1}} S_j \cdot 2^{-(t-T_j)/H_j}\,dt.$$

If one is interested in the average risk contributed by the j-th notification in the time interval $[t_{i-1}, t_i]$, one gets

$$\begin{aligned} &\frac{1}{t_i - t_{i-1}} \cdot \int_A^B S_j \cdot 2^{-(t-T_j)/H_j}\,dt \\ = \; &\frac{1}{t_i - t_{i-1}} \cdot S_j \cdot \frac{H_j}{\ln 2} \cdot \left(2^{-\frac{A-T_j}{H_j}} - 2^{-\frac{B-T_j}{H_j}}\right) \end{aligned}$$

where $[A, B] := [t_{i-1}, t_i] \cap [T_j, T_{j+1}]$.

To conclude, the dynamic risk is obtained as follows:

1. Pick a time line (e.g. logarithmic) $t_1, t_2, \ldots, t_n$.
2. For each t_i, retrieve the dynamic parameters $p(t_i)$, $\mathbb{L}(t_i)$, and $\mathbb{I}(t_i)$ for the given time t_i.
3. Deduce the matrices $\mathbb{P}(t_i)$, $\mathbb{L}(t_i)$, and $\mathbb{I}(t_i)$ according to Section 4.4.
4. Deduce the risk $r(t_i)$ of the entire organisation (or of a specific scenario) for the given time t_i, according to Algorithm 9.
5. Add the point $(t_i, r(t_i))$ to the plot that depicts the risk evolution.

The result of this algorithm for a logarithmic time line is shown in Figure 4.9.

4.7 Conclusion

Deploying security appliances is only one side of the coin. Since incidents can never be fully avoided, it is at least equally as important to properly handle breaches when they occur. In order to be maximally effective, proper incident response relies on knowledge of what happened where – and this information needs to be made available to the security personnel in charge as quickly as possible. Dynamic risk management is one promising candidate to gather all the relevant information in one spot, allowing risk assessors to fall back upon (possibly technical) real-time information from the underlying IT infrastructure.

This chapter presented a strategy for rendering risk assessments dynamic. While hardly any assumptions have been made on the used risk methodology, the dependency model developed in Chapter 2 serves as basis for the entire dynamic risk management process. Moreover, the framework is illustrated with a concrete implementation of such a risk monitoring platform, namely TRICK Service and its API.

In this chapter, the notion of risk was equipped with a decay over time, so that risk monitoring agents cannot only specify the current risk, but also give a hint at the evolution of the risk situation in the near future. That way, the criticality of an otherwise technical alert can be judged better by security managers. Moreover, such data permits to gain an insight on the historic evolution of risk and the quality of incident response management for encountered security issues.

The previous sections have only described how risk information from low-level appliances can be provided to the risk monitoring platform, and how it can be expressed in high-level notions of risk. So far, however, very little attention has been drawn to these agents themselves. The next and final chapter will, in contrast, focus more on the collection of risk information on the field. Apart from discussing how existent monitoring tools can be turned into agents for reporting risk, the major contribution of Chapter 5 will be the presentation of a novel network intrusion detection system. The latter is able to detect a particular family of advanced persistent threats that target static networks, such as those encountered in industrial environments.

Chapter 5

Risk agents

5.1 Introduction

Security appliances are in charge of protecting a system – be it anti-virus solutions blocking malicious files, firewalls preventing externals from accessing internal services, or automatic updates fixing vulnerabilities.

However, none of them are perfectly accurate. To begin with, threats are often very complicated, complex, and diverse, so that it is virtually impossible for these appliances to cover all possible cases of malware and attacks. Second, once defensive mechanisms have been published and deployed, hackers have all time on Earth to find new methods of bypassing, tricking, or disabling them. The former can thus not be solely relied on; security managers need to take several sources of information into account, and have a good sense of security. Finally, humans make mistakes, on several levels. Developers of the security appliances may accidentally introduce flaws. Technical assistants who put these tools into place may configure them in a wrong way. And administrators, who should keep them under surveillance, may not do so out of laziness, lack of time, or lack of knowledge.

Either way, it is crucial to not blindly rely on hardware and software solutions to protect an organisation. While they definitely increase the level of security, humans shall make sure that they actually behave as they should. Therefore, it is a good practise to regularly review log files of these appliances in order to detect suspicious behaviour – both in terms of misconfiguration and possible attacks.

Reviewing log files remains a manual task, however. And this is rightfully the case, since the point of these audits is exactly that one cannot entirely trust decisions made by computers. However, software can help security managers to access the relevant information in an easier way. This is the purpose of the risk monitoring platform, and the *raison d'être* of the risk agents, some of which will be presented in this chapter.

5.1.1 Motivation

The previous chapter presented a risk monitoring platform that receives low-level and technical information from the field, and expresses it in terms of risk. It makes use of the dependency-aware risk model developed in

Chapter 3. However, the latter platform relies on the fact that there exist probes which report relevant information – without the latter, the monitoring platform serves no purpose.

Unfortunately, the information that is made available by security appliances is often technical and specific to the environment. Therefore, additional efforts have to be made to interpret alerts in the context of the entire organisation, and to match it with a concrete risk.

As a simple example, an anti-virus solution raises an alert when it detects a file that is known to be malicious. Since the anti-virus does not typically provide more information on the damage caused by the malicious file, it is not trivial to determine the exact risk which the organisation was exposed to. Only by inferring the type of malware (e.g., ransomware, phishing, trojan, keyloggers) from its name, one can provide more information to risk assessors. Moreover, dependencies will help in determining the extent to which the detected piece of malware could have infected other machines and have thus spread in the organisation.

5.1.2 Objective

With the help of three concrete examples, this final chapter describes how risk can be inferred from the low-level output of security appliances. In particular, it addresses the question what consequences an alert has on the entire organisation in terms of risk. The reflections are then used to describe a procedure for deducing risk levels from the alerts and logs in real time.

The second objective of this chapter consists in providing guidance on how to implement a risk agent for a concrete security appliance. Since it is virtually impossible to cover all possible tools, this thesis restricts itself to three candidates that measure three different kinds of risk.

The first example, an intrusion detection system, determines the risk that the internal network is compromised. It applies machine learning techniques to detect deviations from the 'normal' behaviour of the devices in a network.

The second agent processes log files of a firewall, and thus assesses the risk of external threats. These threats are linked most notably to availability issues, and cover denial-of-service attacks, bot nets, and port sniffing.

The last candidate is given by a tool that analyses the level of vulnerability of the overall network. To do so, it retrieves the list of installed software and compares them against publicly available vulnerability databases.

5.1.3 Outline

This chapter presents several risk agents, each in a dedicated section.

- In Section 5.2, an intrusion detection system (IDS) is presented, which is able to detect a particular family of advanced persistent threats, namely the so-called training attack.

- Since intrusion detection systems only monitor the internal network, typically the only way of assessing external threats (such as distributed denial-of-service attacks) consists in monitoring the firewall. Section 5.3 describes a procedure that allows inferring the risk level from log files produced by firewalls.
- While the previous sections focused on agents measuring the exposure to threats, Section 5.4 presents an utility that assesses the risk originating from vulnerabilities. It relies on package managers and publicly available vulnerability databases.
- A conclusion is drawn in Section 5.5.

5.2 Intrusion detection system

The mission of intrusion detection systems (IDS) consists in spotting security breaches inside the monitored system (which may be a network, an industrial component, or a computer). In contrast to other appliances like firewalls who defeat against external threats, the purpose of an IDS is to detect threats that have compromised (or are about to compromise) the *internal* system.

Therefore, intrusion detection systems are often deployed as secondary auxiliary that makes sure that the primary defences are effective. As such, they are perfect candidates for measuring the residual risk originating from internal threats.

5.2.1 Choice of strategy

Intrusion detection systems can operate in two fundamentally distinct ways [42]: those that rely on a black list of undesired content (called *signature-based* solutions), and those that inspect a network, learn the normal behaviour, and detect any deviations from the latter.

The problem with signature-based systems is the fact that they can only detect *known* malicious behaviour; whenever new attacks become popular, the IDS needs to be updated accordingly. For industrial control systems, this approach is not always viable: often the infrastructure is too critical to allow the installation of automatic patches by an externals (such as the IDS manufacturer). In these scenarios, self-learning appliances are the only option.

However, automated learning systems commonly require a supervised initial training phase, during which it is faced with (manually labelled) benign and malicious data so that it learns the difference between these two data sets. Naturally, for optimal results, the learning process should be carried out directly in the target network, and not in a lab. Nevertheless, many researchers use recorded data sets (such as the KDD'99 [134] data set) to evaluate the performance of their anomaly detection algorithm. Unfortunately, the latter data sets are too generic to be actually used to train and deploy an intrusion detection system in a real network. This common practise can be explained by the fact that the used protocols are often proprietary or unknown, and that the network infrastructure is too complex, undocumented, or not available as a testing environment.

Moreover, an ideal intrusion detection system would spot undesirable content without requiring a training phase, since it can then be directly deployed in any production environment that is not known beforehand. In the machine-learning domain, some schemes already exist which autonomously tell 'normal' data apart from outliers, and which are thus suitable for intrusion detection [98]. For our purposes, clustering-based schemes seem to be the most promising ones, since they are unsupervised, relatively lightweight from a computation point of view (which is important if one wishes to build a real-time intrusion detection system), and allow multiple behaviours to be modelled at the same time (in contrast to Bayesian statistics, which merely splits the data into 'normal' and 'abnormal'). Moreover, they yield comprehensible results, in contrast to e.g. neural networks, where it is not so clear *why* they gave a certain output.

For machine-learning based intrusion detection techniques, a lot of research has been made over the years, that increased their performance, their reliability, and their scope. However, attacks are also becoming more and more sophisticated. The most developed of them are referred to as advanced persistent threats (APT): they cover all kind of hacking or spying activities that are particularly stealthy and persistent [135]. Given the fact that most networks and computer systems rely on anti-virus agents and intrusion detection systems, a lot of money and effort is put now into evading these security mechanisms [135]. Automated learning systems (and especially those that continuously adapt to live data) are particularly affected by this fact, because their learning process can often be manipulated in such a way to make them progressively used to malicious data. This process is referred to as the *training attack*.

It is virtually impossible to design an intrusion detection system that defends against all modes of operation of APTs (and this is especially true when they are targeted, and thus human-operated). Therefore, in order to better understand how stealthy and long-term attacks act on a computer network, this section focuses on a concrete example of an evasion technique that may be used by an advanced persistent threat, namely the training attack. To the best of our knowledge, very little research has been done to date, that analyses the robustness of intrusion detection systems against such evasion techniques.

5.2.2 Threat model

Intrusion detection systems cannot detect all kinds of attacks, and will always make mistakes. Therefore it is important to state precisely which attacks shall be detected by the IDS, and which ones are not actively considered. The purpose of this section is to describe the threat model that is targeted by the intrusion detection system.

First of all, the IDS is meant to be deployed in parallel to signature-based detection engines (such as off-the-shelf anti-virus solutions), and will focus on network attacks, only. Due to the expected mass of data, and due to the increasing use of encryption in network streams, our IDS does not intend to inspect the payload. Instead, it will examine the meta data obtained by modelling the 'behaviour' of the network flows.

Network attacks and induced anomalies

Some network attacks (examples given below) induce a change in the characteristics of a network stream, that otherwise would not occur, or only under rare circumstances. The most prominent example is the distributed denial-of-service (DDoS) attack, which consists in increasing the number of connections to a host, the amount of data sent to a host, the frequency of connecting to a host, or possibly all of them. Note how all of these can be measured independently of the used protocol or of encryption.

Brute forcing is somewhat related, although its goal consists in gaining elevated access rights, rather than bringing down a host. It operates similarly to DDoS attacks, but in a less aggressive way (that is to say, with lower data rates and less connections).

Another good example of abnormal behaviour is network scans. Whereas they do not constitute an attack on their own, they still allow an attacker to make out possible targets and collect information about open ports. In order to launch such a scan, one unavoidably has to make a lot of out-going connections, which are not usually seen on the network.

A fourth category covers all kinds of intrusion and routing attacks. They primarily consist in deviating network streams to third parties ('men in the middle'), who sniff or manipulate the traffic. An IDS could detect such intruders by keeping track of the *connectivity graph*, listing all pairs of machines that are known to communicate with each other. In cyber-physical systems, many processes are regular and repetitive, so watching the timely behaviour of connections might also reveal intruders.

Finally, a less frequent, yet important threat is computer worms. They spread over the network by exploiting vulnerabilities of the host machines. Detecting them is not trivial at all, since the attack pattern depends a lot on the actual exploit; in fact, a single network packet might even suffice, but it will go completely unnoticed in the sheer mass of traffic in busy networks. However, sometimes worms use outdated or unused protocols, the mere presence of which is already suspicious – in that case, a somewhat reliable detection is possible.

All of these threats have good chances to be discovered if one monitors the network flows between every two hosts, and watches out for deviations or sudden changes in their behaviour. That behaviour is characterised exactly by the meta data mentioned above. Following the research of Berthier et al. [137], good candidates for being monitored include:

- the number of bytes transmitted over a certain time period (e.g. 10 seconds);
- the average packet size;

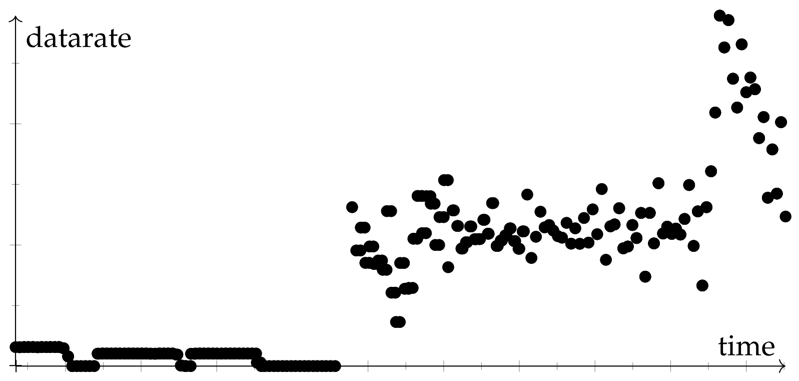


FIGURE 5.1 – Illustration of a typical denial-of-service attack. Note the abrupt 'jumps' for the measured data rate. This example is based on the data sets recorded by Garcia et al. [136].

- the number of concurrent connections;
- the pause since the last packet.

The first two are somewhat related and allow the IDS to detect abuse of a network service; the third one is specifically meant to detect distributed attacks; the last one will aid in finding injected (irregular) packets.

Training attack

Once an intrusion system is in place and learns the typical behaviour of the network, attackers can and will try to evade it. Among the evasion techniques is (what we call) the *training attack*, that is closely related to mimicry and IDS evasion attacks ([45], [138]). In contrast to the latter, it does not only consist in hiding from the IDS, but also manipulates the IDS permanently. It does so by injecting packets that progressively increase any of the monitored quantities, until the target objective has been reached. That way, any future malicious traffic is also considered as normal. See Figure 5.2 for an illustration.

The attack has been thoroughly discussed by Barreno et al. [45], who suggest slowing down the learning process. Although this approach makes a training attack exponentially harder, it will also render the IDS inert. This thesis introduces another strategy that does not suffer from this drawback; it is based on the idea to also consider the long-term evolution of the monitored quantities (see Section 5.2.3).

Stealthy training attack

The training attack consists in slowly shifting the network behaviour towards a malicious state. While these progressive and slow changes are barely noticeable in a short term, they become visible when looked at on a large time scale. However, in order to cover up the training attack even in this scenario, an intruder can hide the malicious traffic by accompanying

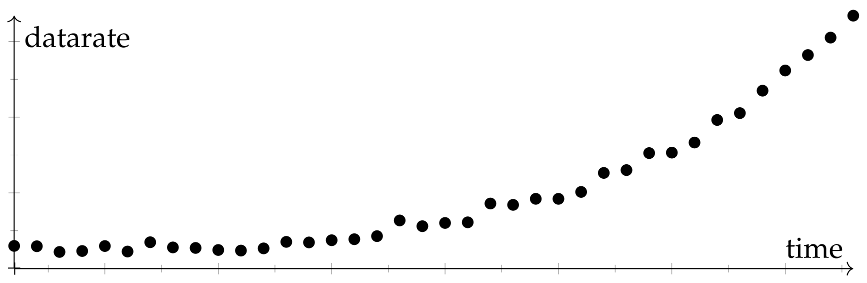


FIGURE 5.2 – Illustration of the training attack over time, for the case of the data rate. An attacker proceeds by progressively injecting more and more packets until he eventually reaches the desired critical threshold.

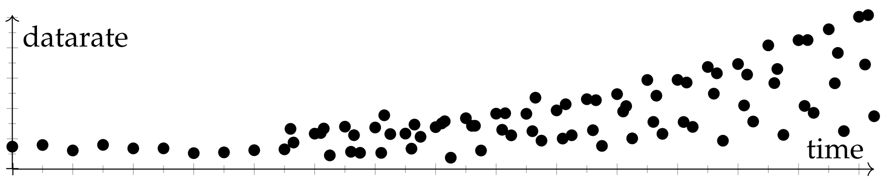


FIGURE 5.3 – Illustration of the *stealthy* training attack over time. Instead of only increasing the traffic load, an attacker creates enough 'normal' data in-between, which outweighs (and thus hides) the malicious traffic.

it with additional, but normal data. So instead of shifting the behaviour *towards* a bad state, he increases the spectrum of behaviours to additionally *include* the bad state. Figure 5.3 illustrates this.

5.2.3 Detecting the training attack

Several techniques have been proposed for detecting attacks in computer networks. Section 2.3 gives an overview of the state-of-the-art intrusion detection schemes.

Threshold and metric based strategies

Among them, the most simple strategy is to fix a threshold for the monitored quantities in advance. It is to be noted that this approach is not based on machine-learning, and thus requires a human to define the threshold(s) for each and every flow in the network. Moreover, such a system is completely inert to changes in the network behaviour, so the thresholds need to be continuously reviewed. This, however, makes the IDS also insensitive to the training attack.

Instead of fixing the thresholds in advance, one could also learn them with the aid of statistical quantities like average and variance. This solution is not as simple as it seems, because it is not so clear what the precise threshold should be. If one assumes that the network load follows a probabilistic distribution, then one can compute the probability that a monitored quantity is according to the probabilistic law, and conversely, if it deviates too

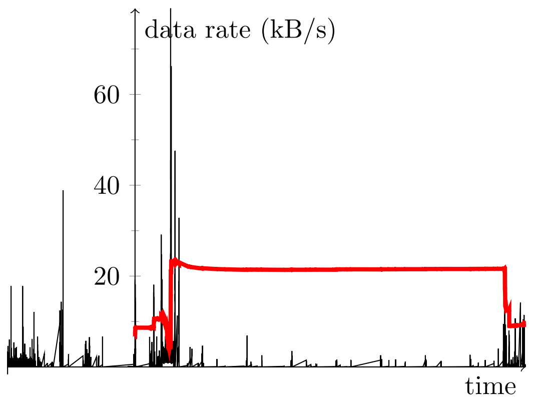


FIGURE 5.4 – Illustration of typical data rates for HTTP traffic (thin black line). Note how the $\mu + 3 \cdot \sigma$ threshold value (thick red line) is not a good descriptor of 'normal' traffic and thus a bad candidate for detecting outliers in the traffic. This is due to the fact that HTTP traffic is not even closely normally distributed.

much. For instance, for a normally distributed quantity $\mathcal{N}(\mu, \sigma^2)$, 99.7% of all samples lie within $\mu \pm 3\sigma$.

Threshold-based approaches suffer from several drawbacks. For one, they do not behave well with inhomogeneous traffic. Indeed, the data rate of a network flow is typically characterised by (at least) two states: an idling state where no communication is made (so data rate 0), and an active state (with data rate δ). Computing the statistical properties of these two states, one ends up with an average data rate of something in-between, which does not yield the desired threshold at all. Experiments suggest that the situation gets worse if more than two states are involved: Figure 5.4 depicts the outcome of such an experiment with real-world HTTP traffic (taken from [139]).

Other machine-learning approaches (including support vector machines and regression methods) that reduce the data to a single metric value suffer from the same issue, for the same reasons.

Second, like any criterion that involves the standard deviation σ of statistical data, the strategy adopted here will encounter stability problems if the traffic approaches constant behaviour (in which case the standard deviation σ is close to zero). One can circumvent the issue by assuming a minimal value σ, but then one has to fall back upon a hard coded parameter again.

Third, this approach is vulnerable to the training attack: even when an attacker uses the linearly increasing sequence of data points $\delta_t := a \cdot t$ (for any constant $a > 0$), a threshold-based IDS will not detect the attack. The proof requires some tedious calculations and is therefore omitted, but one can readily deduce from the definitions

$$\mu(t) = \sum_{i=t-T}^{t} \delta_i \qquad \sigma^2(t) = \frac{1}{T} \cdot \sum_{i=t-T}^{t} (\delta_i - \mu)^2$$

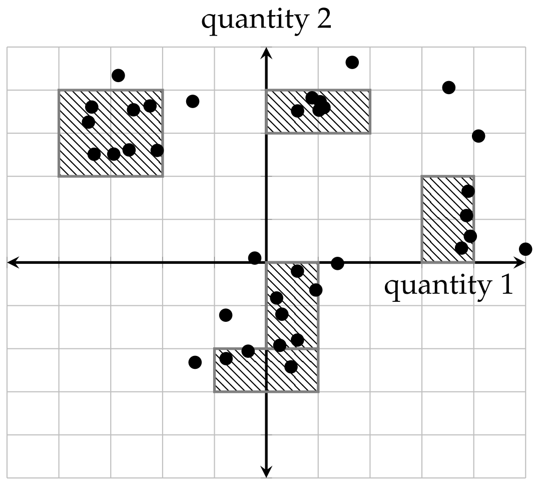


FIGURE 5.5 – Example of a two-dimensional state space divided into a grid, at a particular time t. Bullets denote data points; dense cells are hatched. In this example, there exist 4 clusters at time t. Quantity 1 and 2 could be the traffic rate (in bytes per second) and packet size (in bytes per packet), for example.

that indeed $\delta_{t+1} < \mu(t) + 3 \cdot \sigma(t)$ for all t, so the IDS does not raise an alert at any time. Here, T denotes the number of elements covered by the mean and variance; its value does not have an influence on the statement.

Stream-clustering based strategies

In contrast to many other machine learning techniques, stream clustering algorithms support inhomogeneous traffic quite well, since they particularly *aim* at learning the different behavioural classes: from a sequence of data points, they group similar (or *close*) ones together into a cluster.

Researchers have already put a lot of effort into designing stream clustering algorithms [140] that categorise data streams in real-time. The algorithm that fits the needs of our set-up best is *D-Stream* [141], since it is parameter-less and supports arbitrarily-shaped clusters. It proceeds by dividing the state space into a grid, and continuously computes the density of encountered data points per grid cell. A cell is called *dense* if it contains a certain number of data points; clusters are then defined to be the connected components of dense cells. Figure 5.5 shows an example of such a grid; further details can be found in the original paper by Chen et al. [141].

The clusters account for the several encountered classes of behaviour, while outliers (data points which do not match any cluster) account for statistical abnormalities. It is important to observe that it is not the objective of the IDS to detect the outliers, but changes in the behaviour of the network streams (which manifest themselves by changes in the clusters).

Udommanetanakit et al. [142] have extensively deliberated the possible operations on the set of clusters and their implications. In short, there exist five operations (appearance, disappearance, evolution, splitting, and merging

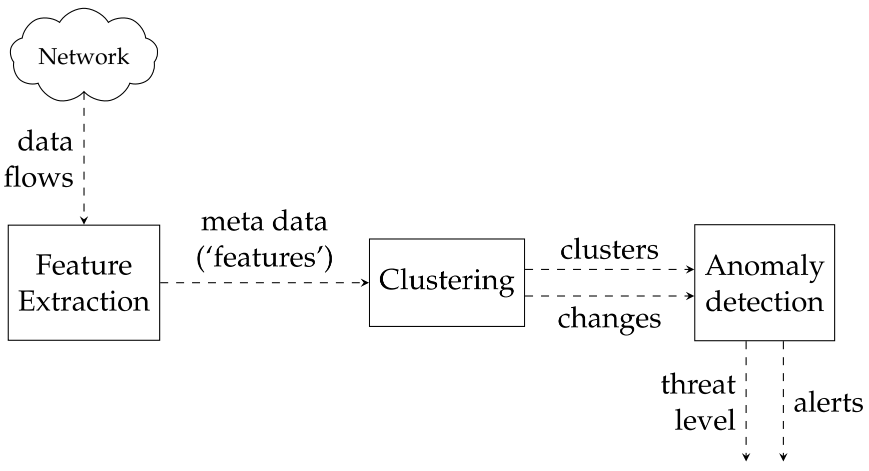


FIGURE 5.6 – Work-flow of the intrusion detection system.

of clusters). Of those, an appearing cluster signals previously unseen behaviour, which indicates an attack or intrusion. Stream clustering algorithms are typically able to detect new clusters in real-time, and are thus good candidates for detecting the network attacks discussed in Section 5.2.2. Regarding the evolution of clusters, the issue is trickier. Although such algorithms account for changes in the input data, they cannot tell small fluctuations (e.g. due to statistical abnormalities) from long-term evolutions of a cluster (e.g. caused by the training attack). So, raising an alert whenever a cluster evolves either yields too many false positives, or forces one to introduce a threshold value (which in turn, requires expert knowledge).

In our work, we improve on existing stream clustering techniques to also detect the training attack, without requiring any additional parameters to be set.

Proposed IDS

The proposed network intrusion detection system (NIDS) builds on top of a stream clustering based IDS. It processes each flow between any two hosts in the networks independently, extracts the relevant features (see Section 5.2.2) and applies clustering techniques to the latter. In this context, a flow comprises all network packets from all data streams sent from the first peer to the second one at a specific TCP/UDP port; for instance, the $\langle 10.0.0.1, 10.0.0.2, 80\rangle$ flow consists of the whole HTTP traffic between the IP addresses $10.0.0.1$ and $10.0.0.2$ (not just one single HTTP connection). Note that the IDS operates completely passively on the whole traffic of a network; thus, it does not interfere with highly critical components of the system, such as industrial control systems.

Figure 5.6 shows the work flow of the IDS. An alert is raised whenever a new cluster is created.

The used data clustering algorithm is based on *D-Stream* [141]. It proceeds by dividing the state space into a grid and keeps track of the density of

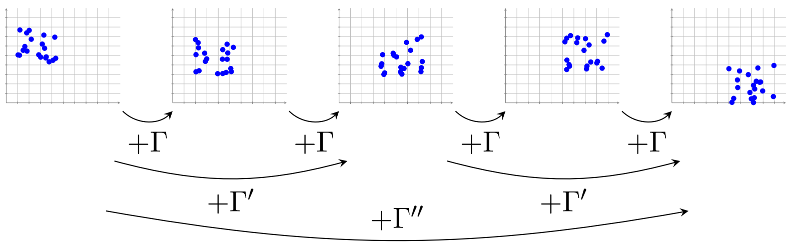


FIGURE 5.7 – The several steps of the training attack and how they appear at different time scales.

recorded data points for each obtained cell. The density includes an exponential decaying over time, which enables the system to adapt to changes in the network. The density $d_C(t)$ of a cell C at time t is defined by

$$d_C(t) = \sum_{p\text{: data point in } C} \lambda^{t-t_p}$$

where t_p is the time when a data point p was recorded. $0 < \lambda < 1$ is called the *decay parameter* and controls how fast the IDS forgets old data. Following the definition in [141], Section 3.2, a cell C is called *dense* at time t if

$$d_C(t) \geq \frac{C_m}{N(1-\lambda)},$$

where N is the total number of cells in the grid, and C_m is a fixed constant. The authors claim that $C_m = 3$ is a good choice; however, experiments (see Section 5.2.5) and calculations (see Section 5.2.4) reveal that $C_m := 10^{-5}$ serves our purposes much better. The huge difference can be explained by the fact that they implicitly assume uniform background noise, whereas our data is locally distributed with almost no noise. Finally, a *cluster* is a maximal connected component of neighbouring dense cells.

The clustering algorithm consists of an on-line phase (during which the densities are continuously updated) and an off-line phase (during which the clusters are updated). In particular, alerts are raised only during the off-line phase, since this is the only moment when new clusters may appear or evolve. The latter phase is triggered at intervals of time Γ. In the following, Γ is referred to as the *clustering interval*.

Detecting the training attack. Deployed as-is, the proposed IDS is still vulnerable to the training attack. This is mainly due to the fact that it can only detect sudden changes at a certain time scale (controlled by the λ and Γ parameters).

However, if several clones of the IDS are launched in parallel, with different λ and Γ parameters respectively, then each instance can detect changes at a different scale. When observed at short time periods, the subtle change in behaviour caused by the training attack may be hard to detect. However, for a different instance that updates its clusters less often, the otherwise slow evolution appears as a deletion together with a creation of a cluster, and is thus detected – see Figure 5.7.

Detecting the stealthy training attack. The classic training attack consists in moving clusters; in contrast, the stealthy version does not touch a cluster, but extends (enlarges) it in such a way that it also includes data points associated to malicious behaviour. Since no new cluster is created, no alert will be raised, either – regardless of the time granularity.

However, what *does* change is the size (number of covered cells) of the clusters. Monitoring those sizes will allow the IDS to detect the stealthy training attack, as well. One could now define a threshold size which a cluster should not exceed, but there is a more elegant way. In fact, the same discussion as the one on intrusion detection techniques above also applies here. So the preferred solution is to apply *yet another* instance of our proposed IDS, but this time on the output (that is, the size of the generated clusters) of the actual IDS. That way, one truly obtains a 'dual' IDS.

Human interaction. Changes of the network behaviour can be natural. For example, if the network topology changes, new machines are connected to the network, or the devices in the network are upgraded, then the behaviour is even expected to change. The intrusion detection system is designed in such a way that it will adapt to the new situation by learning the new behaviour.

However, the same changes of the behaviour could also hint at malicious activity. Even though an alert is raised, the IDS will still adapt to the new situation and possibly miss other malicious behaviour in the future. To prevent it from learning the 'wrong' network behaviour, one could let security managers decide whether a specific alert was legitimate and thus, whether the IDS should adapt to the new behaviour.

Such a strategy has two advantages: it first forces security managers to review alerts; and second, it eliminates possible sources of errors for future alerts, thus limiting the false positive rate. While the prototype IDS *does* present such functionality, the human–machine interaction is not further analysed in this thesis.

5.2.4 Choice of parameters

Decay parameter

The decay parameter $0 < \lambda < 1$ influences the learning capabilities of the IDS. The higher its value, the longer a data point will be 'present' in the grid cell. Recall that the weight of a data point within its cell decays exponentially with time ($t \mapsto \lambda^{t-t_0}$). If $\lambda = 0$, learning will be completely disabled. Setting $\lambda = 1$ will give every data point ever recorded the same weight, so the IDS will always take the whole history into account when computing the clusters, and never forget anything.

The exact choice of the parameter depends on the environment where the IDS is deployed. In an ideal world, where the infrastructure and behaviours remain constant, one would indeed set $\lambda := 1$. Real-world set-ups are different; devices may join or leave the network, software updates may add new functionality yielding a change in behaviour of the network streams, and so forth.

In order to determine a good value for λ, one needs to take into account the estimated period T between (expected) significant changes in the network. Data older than T should rightfully be 'forgotten' by the IDS. For cyber-physical systems, this can be as large as three months. For home networks, several days would be a more reasonable value. Proposition 5.1 provides a maximal value for λ given T, which will be picked in the implementation of the IDS for optimal results.

In the following, we deem that if a data point has $\leq 1\%$ of the total weight available, it is barely noticeable.

Proposition 5.1. *Let $T > 0$. Let N be the number of grid cells. Suppose the data points are recorded in regular time intervals. If $\lambda \leq 10^{-2/T}$, then all data points older than T make up, in total, $\leq 1\%$ of the whole weight available in the grid.*

Proof. Denote the length of the regular time intervals by f. The cumulative weight of all data points ever recorded is

$$\sum_{i=0}^{\infty} \lambda^{i \cdot f} = \frac{1}{1-\lambda^f}.$$

Similarly, the cumulative weight of all data points older than T is

$$\sum_{i=T/f}^{\infty} \lambda^{i \cdot f} = \sum_{i'=0}^{\infty} \lambda^{\left(i'+\frac{T}{f}\right) \cdot f} = \lambda^T \cdot \sum_{i'=0}^{\infty} \lambda^{i' \cdot f} = \frac{\lambda^T}{1-\lambda^f}.$$

By assumption, $\lambda^T \leq \frac{1}{100}$, which concludes the proof. □

Clustering interval

The clustering interval parameter controls how often the data is clustered and scanned for intrusions. If the interval is chosen too long, short-term attacks go entirely unnoticed, since their footprint fades out before the clustering takes place. If it is too short, the engine is unable to detect long-term evolutions.

The strategy is thus two-fold. On the one hand, one needs to choose the longest possible interval Γ so that all short-term attacks can still be noticed. On the other hand, additional instances of the IDS (running with larger clustering intervals, see Section 5.2.4) will make sure that long-term evolutions will not be missed, either. Proposition 5.2 gives a hint on how to choose Γ for our purposes.

Proposition 5.2. *Let $0 < p < 1$. Let N be the number of grid cells. Suppose the data points are recorded in regular time intervals. Assume $\lambda = 1 - \varepsilon$ for some $\varepsilon \ll 1$.*

Set $\Gamma := \frac{N}{p}$, then any data point younger than Γ makes up at least a p part of the weight gained by a cell on average during time Γ.

Proof. Denote the length of the regular time intervals by f. Then $n := \frac{\Gamma}{f}$ data points have been recorded after time Γ. Their cumulative weight is then

$$\sum_{i=0}^{n-1} \lambda^{i \cdot f} = \frac{1-\lambda^{f \cdot n}}{1-\lambda} = \frac{1-\lambda^{\Gamma}}{1-\lambda} = \frac{1-\exp(\Gamma \ln \lambda)}{1-\lambda}.$$

If $\lambda \approx 1$ then $\ln \lambda \approx 0$ and thus $\exp(\Gamma \ln \lambda) \approx 1 + \Gamma \ln \lambda$. Further $\ln(\lambda) = \ln(1-\varepsilon) \approx -\varepsilon$. Using this, one gets

$$\frac{1-\exp(\Gamma \ln \lambda)}{1-\lambda} \approx \frac{-\Gamma \cdot \ln \lambda}{1-\lambda} \approx \frac{\Gamma \cdot \varepsilon}{\varepsilon} = \Gamma.$$

Thus, the average weight gained by a cell is $\approx \frac{\Gamma}{N} = \frac{1}{p}$. The proportion of the weight of any data point younger than Γ with respect to the average is at least

$$\frac{\lambda^{\Gamma}}{1/p} \approx \frac{1}{1/p} = p.$$

□

Proposition 5.2 reveals that if the clustering interval $\Gamma = 600$s is chosen and the state space is divided into $N = 50$ cells, then all data points recorded between two off-line clustering processes will make out $\frac{\Gamma}{N} \approx 8\%$ of the average weight gain of a cell. That is, each of those data points still has a considerable impact on the weight of the cell.

Our experiments also confirm that $\Gamma = 600\text{s} = 10\text{min}$ is a good choice, see Section 5.2.5.

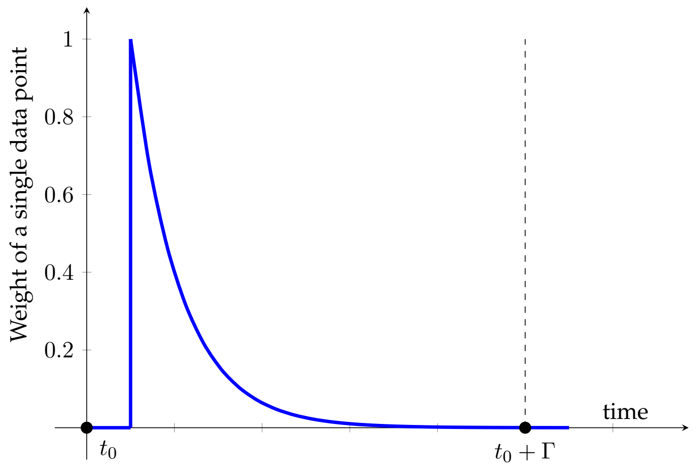
Density parameter

A cell C is defined to be dense if $d_C(t) \geq \frac{C_m}{N(1-\lambda)}$. According to [141], a good choice is $C_m := 3$, but this leads to no cell ever being marked as dense in our set-up. This can be explained by the fact that they have much more data points spread over the whole state space, whereas we deal with locally condensed data. Proposition 5.3 suggests better values for C_m.

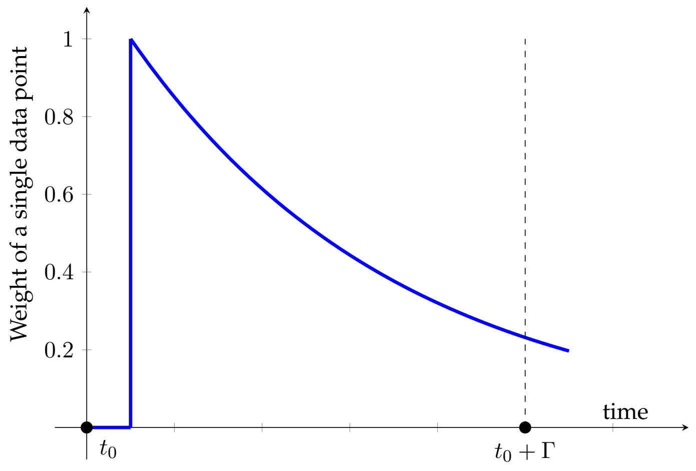
Proposition 5.3. *Let $\Gamma > 0$ and $C_m > 0$. Let N be the number of grid cells. Suppose the data points are recorded in regular time intervals. If $C_m \leq 1 - \lambda^{\Gamma}$, then any cell that received at least the average weight during time Γ, is dense.*

Proof. Denote the length of the regular time intervals by f. Then $n := \frac{\Gamma}{f}$ data points have been recorded after time Γ. Their cumulative weight is

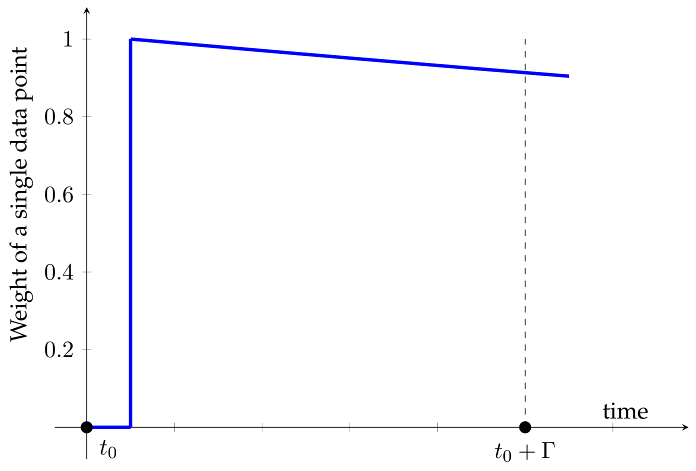
$$\sum_{i=0}^{n-1} \lambda^{i \cdot f} = \frac{1-\lambda^{f \cdot n}}{1-\lambda} = \frac{1-\lambda^{\Gamma}}{1-\lambda},$$



(A) λ too small or Γ too large: data points can be missed between two clustering times t_0 and $t_0 + \Gamma$.



(B) λ and Γ just right: impossible to miss data points missed between two clustering times t_0 and $t_0 + \Gamma$.



(C) λ too large or Γ too small: data points will not be missed, but distant history is not forgotten fast enough, so that new data points are hidden by old ones.

FIGURE 5.8 – Influence of Γ and λ on the weight of a single data point over time.

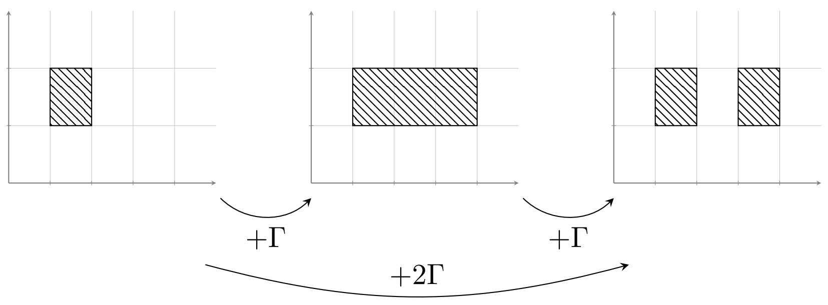


FIGURE 5.9 – The two (legitimate) steps that the training attack consists of. Dense cells are hatched. If the intermediate clustering is omitted, the situation will look as if a new cluster is created, and an alert will be raised.

so the average weight of a cell is $\frac{1-\lambda^\Gamma}{N\cdot(1-\lambda)}$. If a cell C received at least the average weight, then

$$d_C(\Gamma) \geq \frac{1-\lambda^\Gamma}{N\cdot(1-\lambda)} \geq \frac{C_m}{N\cdot(1-\lambda)},$$

so by definition, C is dense. $\square$

For instance, if $\lambda = 0.01^{1/1\mathrm{d}}$ and $\Gamma = 10\mathrm{min}$, then $C_m := 0.03$ would be a good choice. If $\lambda = 0.01^{1/30\mathrm{d}}$ and $\Gamma = 10\mathrm{min}$, then opt for $C_m := 0.001$.

Number of instances

To the clustering-based IDS, the classic training attack as described in Section 5.2.2 appears initially as an enlargement of the cluster, followed by a splitting into two clusters (see Figure 5.9). Note that these are two legitimate steps, so the IDS will not raise any alert.

However, if the off-line clustering between those two steps were omitted, the single resulting step would be correctly identified as malicious, since a new cluster is created. The strategy is thus to apply an additional, independent clone of the IDS with clustering interval $2\cdot\Gamma$. Similarly, the argumentation can be applied recursively on the second IDS to require a third one with clustering interval $4\cdot\Gamma$, and so forth. Eventually they will cover a time span $2^n\cdot\Gamma$ which is so long that an attacker will not bother trying; for instance, if Γ equals 1 month, and 50 years should be covered by the IDS, then one requires $\log_2\left(\frac{50y}{1m}\right) \approx 9$ instances.

The final intrusion detection system thus consists of (for instance) 9 independent instances, each invoked with a different value for Γ. An alert by any of these clones then results in an alert by the final IDS. Since the clones are independent, they can be run on different CPUs or even on multiple devices – no synchronisation is necessary. However, in order to avoid that multiple clones are triggered by the same attack, one can account for only one alert within a given time interval.

5.2.5 Evaluation

We ran several different simulations on various data sets, including [139], [143], and our own recordings of network traffic in an office network. For all experiments that follow, the state space (describing the data rate) is divided into a logarithmic scale of $N = 50$ values ranging from 100 B/s to 10^9 B/s.

Detecting network attacks

The objective of the first set of experiments is to verify if the proposed IDS is actually able to detect classic network attacks (such as denial-of-service). As argued in Section 5.2.4, using a decay parameter of $\lambda := 0.01^{-10\,\text{days}}$ guarantees that any data encountered will be forgotten after roughly ten days. For the clustering interval, we choose $\Gamma = 10\,\text{min}$.

Figure 5.10 depicts the simulation results on the data set [139], which contains 24 hours of network traffic produced by personal computers. As can be read off from the figure, the traffic is relatively low most of the time, and sporadically features small peaks (e.g. (1)–(4) and (8) in Figure 5.10). However, most importantly, the data sets also contains peaks (see (5), (6) and (7) in Figure 5.10) that may be worth being investigated. The objective of the intrusion detection system is to be able to detect this kind of change in behaviour.

Launching a prototype implementation of our IDS (written in Javascript for NodeJS) on the data set [139], we notice that at first, several clusters were created at the very beginning (which acts as a learning phase, so the alerts were expected). The first major peak ((5) in Figure 5.10) is correctly identified as new behaviour, and so is the second one (6). The third peak (7) is classified to be in the same cluster than the previous one, so no alert is raised. Any later traffic matches the expectations of the IDS and does not lead to the creation of new clusters.

If the decay parameter is too low, clustering information will be forgotten too fast. Setting $\lambda := 0.01^{-1\,\text{day}}$ will cause the IDS to forget data after roughly one day, which is also what is observed in the 24-hour simulation depicted in Figure 5.11. In fact, this experiment proves the soundness of the discussion on λ conducted in Section 5.2.4.

Detecting the training attack

We were unaware of any data set that has recorded a training attack, and thus had to create one on our own. For this purpose, the '4SIC Geek Lounge' data set [144] describing 18 hours of real SCADA traffic has been extended (by looping it) to a one-year period so as to obtain a good basis of 'regular' data. Figure 5.12 shows the content of the data set. Note that the two types of behaviour clearly prove the applicability of our intrusion detection approach.

The IDS with parameters $\lambda = 0.01^{-1/7\text{days}}$ and $\Gamma = 10\text{min}$ successfully learns the behaviour of the traffic, raising no alerts. Experiments show that

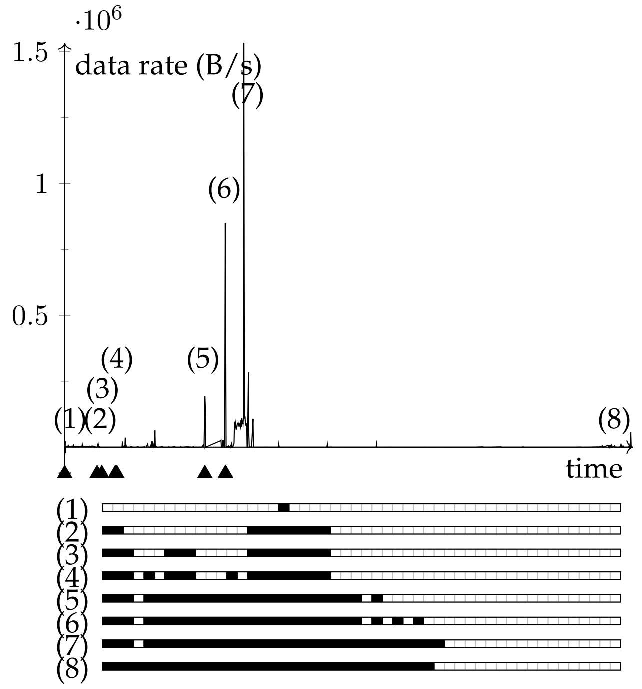


FIGURE 5.10 – Successful detection of unusually high outgoing traffic for one of the hosts in the [139] data set. The triangles denote the times when new clusters were created (thus alerts raised). Below the figure, the clusters are explicitly drawn for the snapshots (1) to (8): boxes represent the cells in the one-dimensional state space; black boxes denote dense cells. After the initial learning phase (1)–(4), a new cluster is created at time (5), and an alert is raised. At time (6), two further isolated clusters are created, and a further alert is raised.

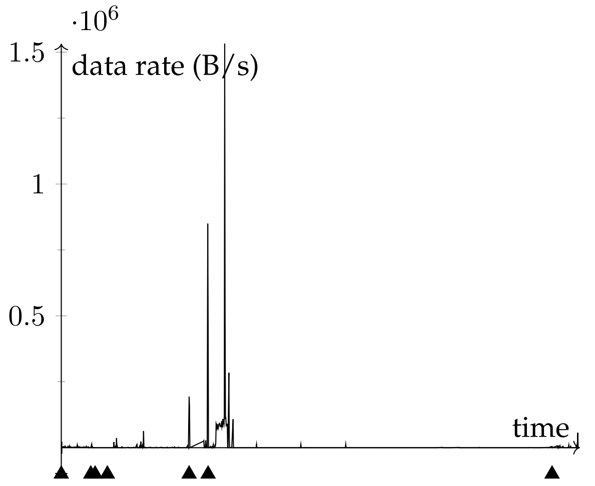


FIGURE 5.11 – A small decay parameter ($\lambda = 0.01^{-1\,\mathrm{day}}$) makes the IDS forget data too fast. The triangles denote the times when new clusters were created (thus alerts raised); note the additional final clustering at the very right of the figure which occurs after approximately a day.

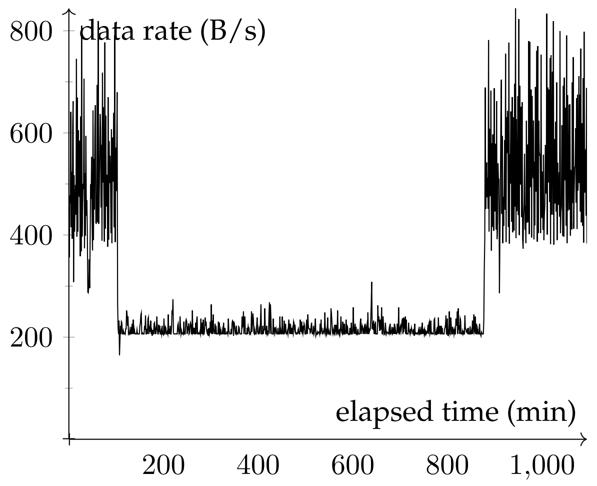


FIGURE 5.12 – The '4SICS Geek Lounge' data set recorded on real SCADA equipment for 18 hours, starting at 5:52 p.m. One can clearly distinguish the night as a period of low activity (7:42 p.m. to 8:30 a.m.).

$\lambda = 0.01^{-1/1\text{day}}$ is too low (it yields false positives) and $\lambda = 0.01^{-1/30\text{days}}$ turns out to be too long in the following discussion (the training attack *does* get detected, but only after almost one year).

In parallel, we crafted artificial data packets with a slowly increasing packet size and frequency, and merged the obtained packets with the real data set. The resulting data set contains the recording of an artificial, yet theoretically feasible training attack. The caused increase in the data rate roughly follows an exponential law ($t \mapsto 1.1^t$).

Following the discussion in Section 5.2.4, several IDS instances are launched on that data set, each of which has a clustering interval twice as large as the previous one. In this use-case, it results in 16 instances ranging from $\Gamma_1 = 10\text{min}$ to $\Gamma_{16} = 228d$. As expected, the first few instances are unable to detect the training attack. However, the long-term clones $(\Gamma_{14}, \Gamma_{15}, \Gamma_{16})$ successfully raise an alert after some initial learning phase – see Figure 5.13 for an illustration.

Two similar experiments have been conducted using a linearly increasing data rate, and an initially increasing but eventually stagnating data rate. In all cases, the longer-term instances were able to detect the attack, while the shorter-term ones were not.

Detecting the stealthy training attack

A much more sophisticated version of the training attack consists in continuously injecting the complete spectrum between normal and malicious traffic. To do so, an attacker repetitively increases the data rate until a given threshold, drops back to normal, and slowly increases again (see Figure 5.14). As predicted in Section 5.2.2, simulations show that our IDS never *creates* a new cluster, but increases the size of existing ones, instead; by consequence, no alert is raised and the attack remains undetected.

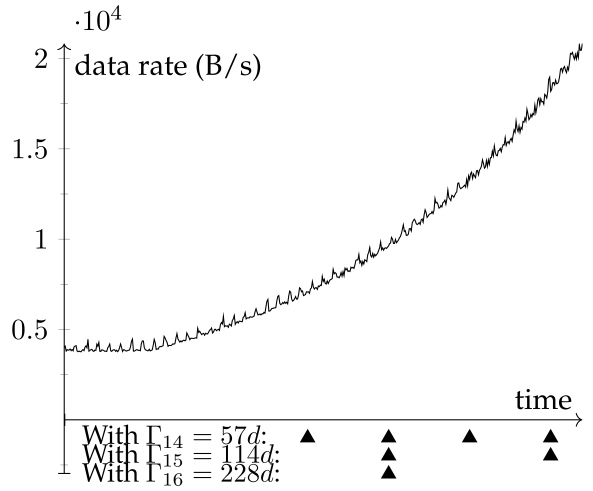


FIGURE 5.13 – The training attack combined with the [144] data set. The triangles denote the times when new clusters were created (thus alerts raised) for the instances with $\Gamma = 57d, 114d, 228d$, respectively.

As a counter-measure, we proposed to include the maximum cluster size in the set of monitored quantities (in parallel with the data rate). In contrast to previous simulations, where the maximum cluster size remained more or less constant (it deviated by at most ± 3), it increased by a decent amount in this case. Indeed, applying the same detection techniques on the maximum cluster size (thus, on the output of the actual IDS) yields the desired results – see Figure 5.14.

5.2.6 Conclusion

This chapter deliberates modern anomaly-based intrusion detection techniques that learn the behaviour of network streams. It is to be noted that the former can only reliably recognise attacks which induce a considerable change in behaviour of the network. Traditional signature scanners perform much better when it comes to detecting specific malware, and should be applied in parallel. While state-of-the-art intrusion detection systems classify individual data packets as good or malicious, the IDS proposed in this thesis rather focuses on grouping similar data packets and decides upon each cluster if it is normal or not. The advantage of this approach is a more stable behaviour with respect to statistical noise, since single outliers do not immediately yield a (false) alert. In addition, the decision of the algorithm can be retraced more intuitively than for other machine-learning based approaches.

This chapter also provides evidence that live learning systems can be tricked by interfering with the learning process, and presents the so-called training attack that makes such a system eventually accept malicious behaviour. We propose a detection scheme that is, to a certain extent, resistant to this kind of attacks. It consists in considering the input at multiple time resolutions, which considerably hardens long-term changes in the behaviour. In order to provider a better understanding of the consequences of a fooled IDS, we

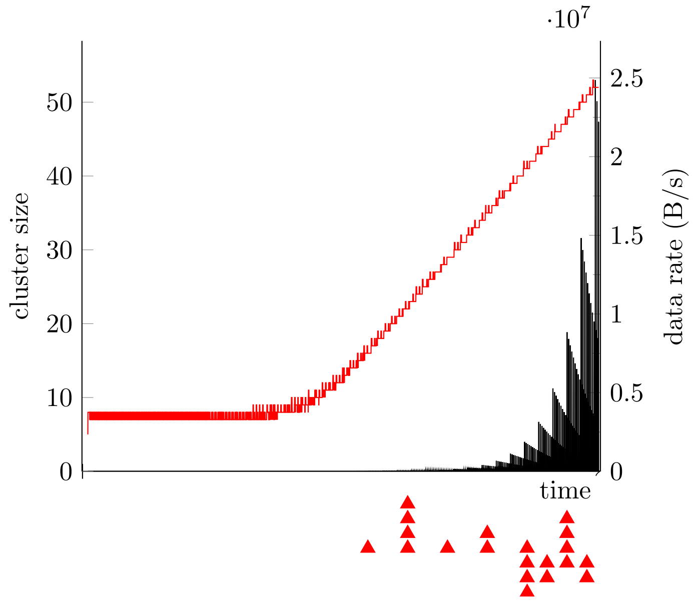


FIGURE 5.14 – The stealthy training attack combined with the [144] data set. The right graph depicts the data rate as induced by the attack, while the left graph depicts the evolution of the cluster size. Although no alert is raised for the behaviour of the *data rate*, the dual IDS does identify the increase of the *cluster size* (alerts marked with red triangles).

presented a stealthier variant of said attack and discussed how to counter it. However, our research only scratched the surface of possible tricking techniques, and there are probably further opportunities for attackers to evade the IDS.

The solidity of our approach is validated, on the one hand, by a mathematically sound choice of parameters, and on the other hand, by simulations conducted on real network traffic from various sources. A prototype implementation has been developed in Javascript for NodeJS, using the libpcap library for capturing the network traffic. Moreover, the IDS has been installed on a Raspberry Pi 3, such that it can be connected to any computer network in a plug-and-play fashion. The latter device was supposed to be evaluated in the test environment of the Luxembourgish smart grid operator; unfortunately, due to delays in the implementation of the latter testbed, no live tests could be made.

5.3 Firewall log parser

Computer networks are often protected with firewalls, blocking illicit access to internal services. In contrast to intrusion detection systems, who monitor the internal network, firewalls see threats coming from the outside. Those attacks include intrusion attempts (e.g. from bot nets), network scans, and denial-of-service attacks. Modern firewalls even provide flood protection against some kinds of *distributed* denial-of-service attacks [145], [146].

Since firewalls are in touch with the threat landscape of the Internet, their log files may give valuable insight on upcoming attacks targeted at the system in question. For example, network scans may indicate reconnaissance activities from hackers, and thus hint at an increased risk of intrusions in the near future.

In this section, two external threats will be discussed: network scans and flooding attacks.

Network scans allow malicious people to identify potential hacking targets. A network scan basically consists in establishing a network connection for all available ports, and in verifying if the targeted server replies; if it does, then there is a corresponding service available at that port.

Flooding attacks, in contrast, aim at disrupting a service. By establishing multiple connections in parallel, hackers try to use up all the resources of a service, rendering it essentially unavailable.

5.3.1 Log files format

To the best of our knowledge, there does not exist any common standard for log files. The approach described in this section will thus slightly differ for other firewall appliances.

The firewall investigated in our use-case[1] provides logs in comma-separated text format, each line representing a connection. Entries are of the following form (expanded to several lines for readability):

```
time=1496221744,
loc=4176575,
fileid=1496181541,
action=accept,
orig=172.16.255.94,
i/f_dir=inbound,
i/f_name=eth0.000,
has_accounting=1,
uuid=<00000000+00000000+00000000+00000000>,
product=VPN-1 & FireWall-1,
rule=12,
rule_uid={00000000-0000-0000-0000-000000000000},
src=10.76.251.12,
s_port=34505,
dst=10.76.251.4,
service=20200,
proto=tcp
```

Independently of the format, log files should feature the following information.

- The time stamp when a connection was recorded (here: `time=`);
- the source IP address (here: `src=`);

[1]FireWall-1/VPN-1 by Check Point.

- the destination IP address (here: `dst=`);
- the destination port, identifying the targeted service (here: `service=`);
- whether the connection is incoming or outgoing (here: `i/f_dir=`);
- whether the connection was blocked by the firewall (here: `action=`).

Source and destination IP addresses identify the remote client and local server, respectively, whereas the destination port represents the targeted software (e.g. '443' for HTTPS server, '22' for remote shell, etc.).

5.3.2 Reading log files

In practise, processing a log file in real-time is not quite as simple as it seems. A monitoring utility needs to periodically check if the file has been changed, and when it has, determine what content is new. This can be time and resource consuming for large files.

Named pipes

Instead of falling back upon files, one can make use of *named pipes*, which are available on all Unix-like systems (such as Linux). Named pipes appear as normal files in the file system, but behave like first-in-first-out (FIFO) queues. Once created, a named pipe can be accessed by two processes: one which acts as a writer, and one as a reader. Content that is written to the pipe by the former can then be retrieved by the latter.

In Linux, a named pipe is created using the `mkfifo` command, for example:

```
mkfifo /var/log/fwlog
```

The configuration file of the firewall then needs to be adapted accordingly, so that the logs are saved at the chosen destination (`/var/log/fwlog` in this example). As for the risk monitoring agent, it opens the named pipe for reading and waits for new content to arrive. The nature of the FIFO queue guarantees that every line is processed exactly once.

Algorithm 10 Reading new content from a named pipe.

```
1: open named pipe for reading
2: for each read line $L$ do
3:     process $L$ (see sections below)
4: end for
```

Manual approach

It may be the case that named pipes cannot be used as described above. This is for instance the case when the path of the log file cannot be configured, or when the logs are split over several files (also called log rotation).

In this case, the monitoring agent needs to be invoked in regular time intervals (e.g. using `cron`). It then verifies each time if the file has changed, and what content has been added. To this end, however, it needs to keep track of the last processed line, and skip all lines that have been processed already. The procedure is described in Algorithm 11

Algorithm 11 Manually reading new content from multiple log files.

Input: identifier i (e.g. timestamp) of last read line

```
1: for each relevant log file do
2:     open log file for reading
3:     for each read line $L$ do
4:         if $id(L) > i$ then
5:             process $L$ (see sections below)
6:         end if
7:     end for
8: end for
```

A slightly more efficient approach can be adopted if all logs entries are saved in a single (eternally increasing) file. Indeed, in that case the monitoring agent only needs to keep track of the file size and jump to the appropriate place in the file. That way, the agent does not have to re-process the entire file each time. Algorithm 12 shows the improved variant.

Algorithm 12 Manually reading new content from a single, eternally increasing log file.

Input: file size s of the log file when agent was last invoked

```
1: open log file for reading
2: jump to position $s$
3: for each read line $L$ do
4:     process $L$ (see sections below)
5: end for
```

5.3.3 Network scans

Network scan are characterised by several attempts to establish a connection on multiple ports. This is in strong contrast to the normal behaviour of a client, who typically uses only one service (i.e., port) at a time. In order to detect network scans, it is thus enough to observe the diversity of incoming connections that have been blocked.

However, the monitoring agent must also be able to 'forget' connections from long ago, because otherwise it can yield false positives. Indeed, suppose a network scanning attack was detected. Then, for every (single) blocked connection in the future, an alert will be raised again, even though the connection is not necessarily part of another network scan. In order to avoid this behaviour, the agent also needs to take the time in account, when the connection was encountered.

In simple terms, a network scan can be defined as the attempt to establish N connections (to N different ports, respectively) within a given time window T. Finding good values for N and T in the definition is non-trivial

[147], however. To detect scans, one keeps track of the last connections; if there are at least N different ports within the last time T, a network scan is occurring (by definition). An alert is raised whenever these N times fall within the interval $[t_{\text{now}} - T, t_{\text{now}}]$. These alerts can then be reported to the risk monitoring platform as described in Chapter 4. Algorithm 13 describes the approach in greater detail.

More advanced algorithms exist that detect network scans more reliably [147]–[149], and also handle stealthy scanners [150], [151]. These are out of the scope of this thesis, though. Moreover, these algorithms typically need more fine-grained data than just firewall logs.

Algorithm 13 Detecting network scans from firewall logs.

```
Input: $N \in \mathbb{N}$   (number of allowed connections)
Input: $T > 0$   (time window during which these connections are allowed)

1:  $t \in \mathbb{R}_+^N$
2:  $p \in \{0 \ldots 65535\}^N$        ' A list of N ports
3:  for $i = 1$ to $N$ do
4:      $t_i := 0$
5:      $p_i := 0$
6:  end for

7:  for each logged connection $c$ do
8:      ' Update the 'last seen' time for the port of this connection
9:      for $i = 1$ to $N$ do
10:         if $p_i = \text{port}(c)$ then
11:             $t_i := \text{time}(c)$
12:         end if
13:     end for

14:     ' Handle over-saturated list of ports
15:     if $\forall i : \text{port}(c) \neq p_i$ then
16:         if $\exists i : p_i = 0 \lor t_i + T < \text{time}(c)$ then
17:             ' Port $p_i$ is no longer used
18:             $p_i := \text{port}(c)$
19:             $t_i := \text{time}(c)$
20:         else
21:             ' Too many ports are simultaneously used
22:             raise alert

23:             ' Replace the oldest connection by this one
24:             $j := \arg\min_{i=1}^{N} t_i$
25:             $p_j := \text{port}(c)$
26:             $t_j := \text{time}(c)$
27:         end if
28:     end if
29: end for
```

Correctness The approach of Algorithm 13 is correct, because if N connections to N different ports have been detected within an interval T, then by definition a network scan is happening, indeed.

Inversely, a network scan implies that there are N connections to N ports within an interval T. It might be the case that there are additional connections *not* part of the network scan, though, possibly on other ports. There are two cases. First, if such an additional connection is encountered on a port that is also used by the network scan, then the respective time t_i in Algorithm 13 may be even more recent. Second, if there are additional connections on ports *not* covered by the network scan, then Algorithm 13 replaces the respective times t_i by more recent ones, as well. In any case, all times t_i are still within a interval T, and the network scan is thus detected.

A final note is to be made on the definition of a network scan. It might be the case that two independent clients each establish $\frac{N}{2}$ legitimate connections on distinct ports, so that Algorithm 13 detects a network scan, even though this was not intended by either client. A server cannot tell the difference though, for it does not know if the two clients are independent or secretly collaborate.

So Algorithm 13 correctly identifies a network scan as it was defined above, even though it might not be an actual network scan. However, if N is decently large, then any activity that involves connections to $\geq N$ distinct ports is suspicious, whether it originates from a scan or not.

Value of N and T The time interval T should be picked in such a way that it covers the time that network scanning tools need to perform a scan. Lee et al. [149] suggest $T = N \cdot 120$ *seconds*, since "most port scanning tools set the time between the packets to be much less than 120 seconds".

The choice of N actually depends on the number of services that are exposed to the public Internet. It should be large enough to allow the distinction between legitimate clients and scanners, but also low enough to minimise the memory use. Data collected by Sridharan et al. [148] states that for typical servers, $N = 3$ is already enough to statistically distinguish between scans and legitimate connections. Indeed, according to the 2006 study, most servers have only one port exposed[2]. The results of the study suggest that setting $N := 1.5 \cdot$ (number of opened ports) yields promising results.

5.3.4 Flooding attack

The flooding attack consists in sending massive amount of data to a server with the intention of exhausting its resources and thus taking down its provided services. The effectiveness of the attack is increased a lot when multiple such attacks are launched in parallel, in which case one refers to a *distributed* denial-of-service (DDoS) attack.

DDoS attacks are still a huge concern for organisations today, as show the reports of recent cases [152]. Detecting and mitigating these attacks are thus of a critical importance, especially for industrial control systems that need to be functional at all time.

[2]It is to be noted that not every server falls under this criterion. Also, with the rise of the 'Internet of Things', the trend is typically towards opening more ports.

(Classic[3]) firewalls are neither particularly secure against these attacks [153], nor can they successfully mitigate them [154]. This is due to the fact that firewalls are typically stateful, meaning that they have to remember the connection state for every client – which becomes problematic when there are suddenly many clients. But firewalls were not designed for this purpose; while their mission is to filter out illicit connections, DDoS attacks typically consist in sending massive amounts of data to an *allowed* port. Nevertheless, firewalls can still *observe* an on-going attack, to the extent that they can handle the traffic.

Since DDoS protection is an entire research topic on its own, that issue is not addressed here. Instead, the risk monitoring agent described in this section will try to detect the occurrence of a flooding attack, and report it to the monitoring platform. That way, it serve two purposes: first, it increases the overall risk of unavailability for the affected assets in real time; and second, it provides evidence for a later investigation, stating why a service went down.

Some research has been conducted for the early detection [155]–[158] and prevention [158] of massive flooding. However, the goal of this section is to describe how existing hardware (such as firewalls) can be turned into risk monitoring agents. Any appliance that more specifically detects flooding attacks will yield more valuable results, of course. And as long as the latter produces relevant output (log files), the concepts developed in this section can be easily adapted to such an appliance as well.

In the context of this section, a very simple approach is adopted, due to the limited information available in the log files, and due to possible constraints on the computing power of the risk monitoring agent. More advanced detection techniques can be used if further information is known on the connections. For example, Lakhina et al. [158] compute the entropy of the transferred bytes, Yu et al. [156] analyse the number and frequency of packets within a connection, Jin and Yeung [155] compare the number of opened connections against the number of closed connections, and Sekar et al. [157] propose a two-staged scheme based on an initial filtering and a final clustering phase.

The proposed approach, given in Algorithm 14, is based on a statistical modelling of the number of connections over time. In contrast to other intrusion detection schemes, this strategy allows the agent to not only raise a deterministic alert, but to provide its belief (in probabilistic terms) that a DDoS attack is being observed.

The algorithm proceeds by chopping the time into intervals of duration $T > 0$, during which it counts the number of connections and thus obtains a certain connection rate per second. The $N_H \in \mathbb{N}$ most recent rates are then kept in a queue H. Each newly measured connection rate δ is then compared to the historical data H to yield the probability that δ matches the statistical behaviour. Mathematically, this expression is given by the error function $x \mapsto \text{erf}(x)$, which is defined to be the probability that a normally distributed random variable $\mathcal{N}(0, 1)$ lies within the interval $[-x, x]$.

[3]Nowadays, commercial firewall solutions exist that additionally provide DDoS protection as an additional feature.

Approximations exist for computing this value numerically (see e.g. [159]). The thus obtained likelihood can then be directly communicated to the risk monitoring platform as described in Chapter 4.

Algorithm 14 Compute likelihood of a distributed denial-of-service attack.

```
Input: Time window size $T > 0$
Input: Size of history $N_H \in \mathbb{N}$

 1: $H$ := empty queue
 2: $t_0 :=$ nil
 3: $n := 0$

 4: for each logged connection $c$ do
 5:     if time$(c) - t_0 < T$ then
 6:         ′ Count number of connections in time window
 7:         $n := n + 1$
 8:     else
 9:         ′ Compute connection rate in time window
10:         $\delta := \frac{n}{T}$

11:         ′ Reset time window
12:         $t_0 :=$ time$(c)$
13:         $n := 1$

14:         ′ Compute average and standard deviation
15:         $\mu := \frac{1}{|H|} \sum_{h \in H} h$
16:         $\sigma^2 := \frac{1}{|H|-1} \sum_{h \in H} (h - \mu)^2$
17:         $p := \mathbb{P}\left[|\mathcal{N}(0,1)| < \frac{|\delta - \mu|}{\sigma}\right]$
18:         report $p$ to risk monitoring platform

19:         ′ Keep track of history
20:         add $\delta$ to $H$
21:         if $|H| > N_H$ then
22:             remove oldest item from $H$
23:         end if
24:     end if
25: end for
```

A Python prototype has been developed that reads log files in CSV[4] format, and that computes the probability according to Algorithm 14.

5.3.5 Conclusion

Even though firewalls were not designed to defend against reconnaissance and denial-of-service attacks, one can still observe the latter in the firewall log files. It is to be noted that more sophisticated (and commercial) solutions exist which detect (and even prevent) the attacks in a more reliable way. However, the objective of this section consisted in showing how *existing* appliances can be incorporated into the risk monitoring system.

[4]Comma-separated values.

5.4 Patch management

A vulnerability can undermine the security of a system, because it causes basic assumptions to no longer hold, which can have devastating consequences. For example, a recently discovered flaw allowed remote attackers to execute arbitrary code on a firewall and thus gain full control over it (CVE-2018-0101[5]). The vulnerability thus does not only expose the firewall itself at risk, but also the entire network behind it.

Is it therefore crucial to patch critical vulnerabilities as soon as possible. Thankfully, on some systems (particularly Linux), updating has become very easy with the rise of package managers. A package manager is a piece of software that allows its users to request the installation, upgrade, patching or removal of software with a single command.

Even though package managers simplify the task a lot, patch management still relies on human operators to check for updates, evaluate their effectiveness and possible side effects, and finally deploy them [160]. In consequence, it may happen that the vulnerability is not immediately closed, and that the system remains exposed to an increased risk.

This section will further analyse the risk resulting from unpatched vulnerabilities. Moreover, it presents an agent that determines the performance of the patch management process from the time that is needed to apply security updates. The latter performance can then be reported to the risk monitoring platform described in Chapter 4, and used as a vulnerability parameter in a risk analysis.

5.4.1 Work flow

The patch management agent proceeds by listing all available updates and by inferring an overall performance indicator for the patch management on the given machine. A more precise work flow is given in Algorithm 15, and the individual steps are described in the sections below.

Although the process described below is system-independent on principle, for the sake of illustration this document focuses on Debian operating systems and the APT package managing utility.

Algorithm 15 Work flow of the patch management agent.

```
1: Obtain a list $U$ of all available updates via apt list --upgradable
2: for $u \in U$ do
3:     Extract the release date $r_u$ from the description of $u$
4:     Fetch the list $E_u$ of CVE information related to $u$
5:     Deduce performance $P_u \in [0, 1]$ from $E_u$ and $r_u$
6: end for
7: Infer overall performance $P := \min_{u \in U} P_u$
8: Report $P$ to the risk monitoring platform
```

[5]https://nvd.nist.gov/vuln/detail/CVE-2018-0101

5.4.2 Criticality of an update

When exploits become known for software packages, its maintainers or independent security researchers publish a notice on the newly created vulnerability, so that every user is aware of the risk that is faced by their system. The *de facto* standard method for publishing such notices is the Common Vulnerabilities and Exposures (CVE) system, maintained by the non-profit MITRE Corporation[6]. MITRE also operates a repository that maintains a list of such CVE reports, which is available at `https://cve.mitre.org/`.

In parallel, the National Institute of Standards and Technology (NIST) maintains the National Vulnerability Database (NVD), which augments the CVE list with additional information. In particular, they provide a standardised method, CVSS[7], for estimating the criticality of a vulnerability with the help of a score, ranging from 0 (not critical) to 10 (very critical). The score of a CVE record can then be retrieved directly from the National Vulnerability Database. The patch management agent will mainly use this information to assess the risk originating from a known vulnerability.

Deciding whether a vulnerability is applicable for a machine is hard, in practise. Indeed, when a vulnerability exists in a auxiliary library, then one cannot know from the CVE description alone whether a certain software application is affected, or not. Moreover, it is sometimes not obvious what software version or edition a CVE report refers to, and if the affected version is installed on the system.

Fortunately, for the case of Debian, these issues have been addressed. More precisely, the Debian security tracker[8] associates CVE information to security patches that are available for Debian software packages. That way, when fetching the list of available updates, one can additionally retrieve CVE information about the associated vulnerabilities, and thus also the CVSS score. This can either be achieved using simple HTTP requests to the security track website, or by using available tools like `debsecan`[9].

5.4.3 Patching deadline

Since risk is an interaction of threat and vulnerability, the mere presence of a vulnerability does not suffice to put a system at risk. Without a corresponding threat (which is a software exploit in this case), no incidents are to be expected. Regarding exploits, time plays an important role for the security of a system, starting at the moment when a vulnerability is disclosed. According to the Microsoft Security Intelligence Report [161], most exploits are observed within the first day of the disclosure of the vulnerability – in which case one refers to a zero-day exploit. Moreover, hardly any exploit is encountered 30 days after the disclosure of a vulnerability.

[6]`https://www.mitre.org/`
[7]Common Vulnerability Scoring System, `https://nvd.nist.gov/vuln-metrics/cvss`.
[8]`https://security-tracker.debian.org/`
[9]`https://wiki.debian.org/DebianSecurity/debsecan`

Rating	CVSS	Patch deadline (in days)
Low	0.1 – 3.9	30 – 8
Medium	4.0 – 6.9	8 – 3
High	7.0 – 8.9	3 – 1.5
Critical	9.0 – 10.0	1.5 – 1

TABLE 5.1 – Qualitative rating of CVSS scale and proposed deadline for applying patches.

It goes without saying that the longer a vulnerability has been known, the higher will be the risk that an exploit exists in the wild. When it comes to evaluate the performance of patch management, it is thus sensible to consider the ages of available patches: the longer they have been available, the more urgently they should be installed.

Given the results from the Security Intelligence Report [161], it is sensible to require that highly critical vulnerabilities be patched the very same day, and that less important security updates be installed within 30 days after they have been released[10]. According to [162], critical vulnerabilities correspond to CVSS scores ranging from 9.0 to 10.0, whereas scores less than 3.9 indicate low severity. Table 5.1 summarises all the reflections above.

There are many ways to interpolate the recommended patching deadline $D(s)$ for intermediate values of the CVSS score s – the most simple one being a linear regression. However, since high-severity vulnerabilities should also be patched comparatively fast, we opted for an exponential regression instead, given as follows.

$$D(s) = \exp\left(-\frac{s}{3}\right) \cdot 30 \text{ days}$$

Note that $D(0) = 30$, $D(4) \approx 8$, and $D(7) \approx 3$ days. In contrast, a linear regression would have yielded $\tilde{D}(7) \approx 10$ and $\tilde{D}(9) \approx 4$, which seems too long for high-severity vulnerabilities.

5.4.4 Performance of patch management

In an ideal world, updates are evaluated and deployed almost instantly after they have become available. This assumption is not realistic, though, because delays can occur at any stage of the patch management process. Brykczynski et al. [160] have identified four phases which the patch management process goes through:

- patch awareness delay, the "duration between the vendor publishing the patch and the organization becoming aware of it";
- patch evaluation latency, the "delay between the organization becoming aware of a patch and beginning the evaluation";

[10]Note that the discussion is about vulnerabilities and security patches, not about feature updates.

- patch evaluation duration, during which the organisation weighs the risk (from side effects) against the benefits;
- patch implementation duration, the "period between the organization becoming aware of the patch and implementing the patch fully".

Depending on the complexity of the patch, each stage can take more or less time. In any case, however, the performance of the entire process correlates with the response time of the people in charge for taking the appropriate decisions.

It is thus sensible to use the latter response time to evaluate the quality of the entire process. In Section 5.4.3, a relation has been determined to deduce a deadline from the criticality of an update. The idea is then to verify if the deadline has been respected, and to judge the performance of the patch management process based on the latter information.

The performance $P(t) \in [0, 1]$ should satisfy the following (natural) requirements:

- When all patches are applied (i.e. no further updates are available), P should be 100%.
- When an update becomes available, P should stay relatively high as long as the deadline is respected.
- After the deadline, P should drop rapidly to a value close to 0.
- When there are multiple updates, P should depend on the most critical one. Indeed, the security of a system is only as strong as the weakest link in the chain [163].

Many curves fit these requirements, so the choice is quite arbitrary. Nevertheless, we try to opt for a comparatively short one, which is continuous over time and simple to compute. First pick two levels Λ and λ, representing the 'relatively high' and 'close to 0' levels from the requirements above. The proposed performance indicator over time, $P(t)$, is then given as

$$P(t) = \min_{u \in U} P_u(t)$$

$$P_u(t) = \begin{cases} 1 - (1 - \Lambda) \cdot \dfrac{t - r_u}{D_u} & \text{if } t - r_u \leq D_u \\ \Lambda \cdot \left(\dfrac{\lambda}{\Lambda}\right)^{\frac{t - r_u}{D_u} - 1} & \text{if } t - r_u > D_u \end{cases}$$

where $D_u = D(s_u)$ represents the deadline as computed in Section 5.4.3 above, and r_u denotes the release time of the update.

The proposed function $P_u(t)$ essentially behaves like a linear function before the deadline, and like an exponential decay after the deadline. It is depicted in Figure 5.15 for $\Lambda = 90\%$ and $\lambda = 1\%$. One can verify that $P_u(D_u) = \Lambda$ and $P_u(2D_u) = \lambda$, and that $P(t)$ indeed satisfies all criteria mentioned above.

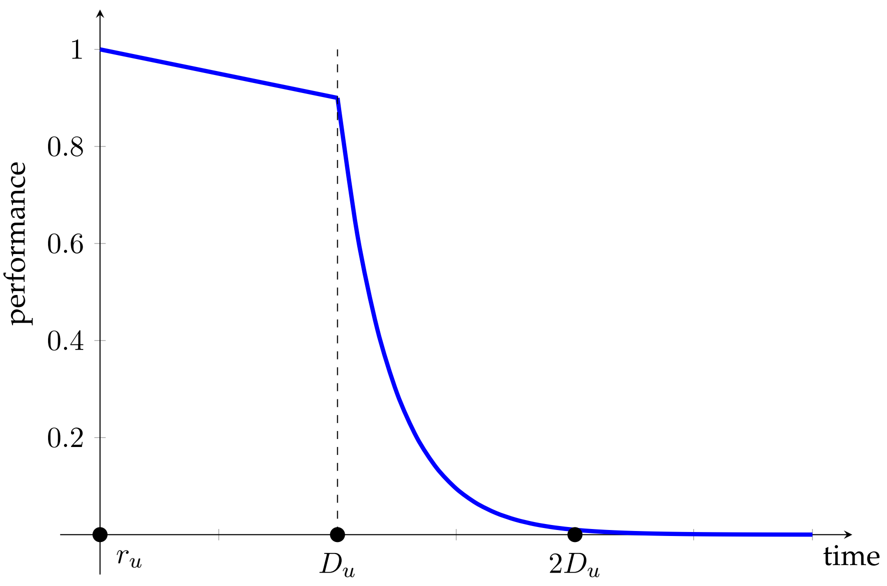


FIGURE 5.15 – Performance indicator over time for a single update.

5.5 Conclusion

In this final chapter, several risk monitoring agents were presented which provide input to the risk monitoring system, and thus the related risk analysis. They measure the risk of the monitored system at several levels:

- Section 5.2 develops an intrusion detection system that detects *internal threats*;
- Section 5.3 presents a firewall log parser which assesses *external threats*;
- Section 5.4 describes an update checking tool that searches for known *vulnerabilities* on Linux servers or workstations.

These agents are in charge of interpreting the technical information (contained e.g. in logs) and convert them to equivalent notions of risk. In the context of this thesis, these notions are expressed in terms of numerical values for the likelihood and impact of risk scenarios. The determined values are then reported to the risk monitoring platform (specified in the previous chapter), which takes care of passing it on to the appropriate risk analysis.

It is to be noted that the firewall and patch management agents have been purposefully designed simple. The objective of these use-cases consisted in showing that any existing device or software can be easily turned into a risk monitoring agent, given that it provides sufficiently relevant information. Naturally, the higher the sophistication level of the appliance, the more accurate and precise the monitored risk will be. This has been shown with the main use-case, an intrusion detection system, that is specifically conceived for detecting unwanted flows within the monitored network.

Chapter 6

Conclusion

6.1 Addressed topics

This thesis deliberated a new approach to dynamically monitor risk for an organisation, focusing on (but not limiting to) industrial control systems. While some risk assessment methodologies exist that address the volatile nature of risk, especially in time-critical environments such as found in the industry, they are often designed for a very specific use-case and do not depict the overall risk faced by the organisation.

The work presented in this thesis is divided into three parts, each of which tackles one of the objectives set in the introduction.

The first objective was to provide an intuitive and easy-to-use framework for modelling the complex structure of large organisations, such as industrial control systems. The thesis identified a need for modelling the interactions between the individual components in terms of risk dependencies. The risk assessment model presented in Chapter 3 does not only provide this possibility, but it is also compatible with a large variety of existing risk management methodologies. Indeed, the dependency graph technique described in Chapter 3 can be applied in parallel to a risk management system, and improves the likelihood estimates that were made manually in the latter.

The second objective consisted in developing a solution for reporting real-time risk information into a risk analysis. While the proposed platform was implemented to a specific risk management tool, TRICK Service, the thoughts from Chapter 4 were purposefully held generic enough so that they can be adapted to other risk analysis methodologies without greater efforts.

The last objective dealt with obtaining the necessary information from technical sources, so as to evaluate the risk 'on the field'. Several agents have been developed in Chapter 5, the most prominent one being an intrusion detection system that specialises on the detection of a certain class of advanced persistent threats (namely training attacks).

6.2 Prospects

Apart from the description of the risk monitoring framework, this thesis allowed the development of several tools that ease (and automate) the work of a risk assessor. Those tools include:

- The risk monitoring platform 'TRICK API' as an extension to the related risk analysis tool 'TRICK Service', that is described in greater detail in Section 4.5.3. The resulting piece of software can be used to include dynamic risk indicators from external agents into a risk analysis. The tool was also developed in the context of the 'SGL Cockpit' project, which consisted in developing a tool suite that dynamically monitors the risk in the Smart Grid of Luxembourg (SGL). It will be deployed into the central system of the Luxembourgish smart grid operator.
- The dependency modelling tool 'DepOT' (presented in Section 4.5.4), that facilitates the creation of dependency graphs for risk assessors. It features a functionality to allows the modelled graph to be imported into the TRICK Service risk analysis tool. Nevertheless, it does not necessarily have to be used within a context of dynamic risk monitoring; instead, it can also be used as a tool to produce purely informative content for traditional risk analyses.
- An intrusion detection system (IDS) that spots intrusions in a network with statistically predictable flows. Such networks can be typically found in industrial control systems which have very little interaction with human users. The intrusion detection system does not only learn the normal behaviour of the network, and raises alerts on deviations from the normal state. It was additionally designed in such a way that the risk originating from certain classes of attacks can be inferred in real time, including but not limited to: denial of service, physical intrusion, sniffing, and the training attack. The IDS is planned to be deployed at the Luxembourgish smart grid operator[1]. Moreover, due to collaborations with the Horizon 2020 project 'ATENA'[2], the IDS will also be integrated into other industrial environments in the electricity, gas, and water distribution sectors.
- Two additional risk monitoring agents, whose purpose mainly consisted in illustrating how existing security appliances can be turned into sources of risk indicators.

The main strength of the outcome of this thesis is the modular nature of the individual achievements. It is true that every tool, including the dependency graph model, plays its part to the entire risk monitoring platform. However, each of them can also be used separately, without depending on the other tools or on the underlying risk management system. As a matter of fact, organisations can choose to only implement that part of the platform which concerns them, or which is sensible to them – a property which is not necessarily guaranteed by state-of-the-art solutions.

[1]In a later stage though, due to delays linked to the implementation of the smart grid infrastructure.

[2]Partially funded by the European Commission, grant agreement number 700581.

Appendix A

Proofs

Proposition A.1. *Let $G = G(V, E)$ be a causal graph. For ease of writing, if $v \in V$ and $x \in \{0, 1\}$ and $\pi : V \to \{0, 1\}$, set*

$$f_\pi(v, x) := \mathbb{P}\left[\mathbb{I}_v = x \,\middle|\, \mathbb{I}_{\mathit{parents}(v)} = \pi|_{\mathit{parents}(v)}\right].$$

Then

$$\sum_{\substack{\pi:V\to\{0,1\}\\ \pi(\alpha)=1}} \prod_{v\in V} f_\pi(v, \pi(v)) = \sum_{\substack{\pi:V\to\{0,1\}\\ \pi(\alpha)=1}} \prod_{v\in \mathit{parents}(\alpha)} f_\pi(v, \pi(v)).$$

Proof. Let $G = (V, E)$ be a causal graph. Since G is in particular directed and acyclic, there is an ordering of the vertices such that the parents of every vertex v' come after v'.

Fix $\alpha \in V$. Let w be the first vertex in that ordering which is not an ancestor of α, so by definition it cannot have any children. As such, $f_\pi(v)$ only depends on w if $v = w$. Hence,

$$\begin{aligned}
&\sum_{\substack{\pi:V\to\{0,1\}\\ \pi(\alpha)=1}} \prod_{v\in V} f_\pi(v, \pi(v))\\
&= \sum_{\substack{\pi:V\to\{0,1\}\\ \pi(\alpha)=1}} f_\pi(w, \pi(w)) \prod_{v\in V\setminus\{w\}} f_\pi(v, \pi(v))\\
&= \sum_{\substack{\pi:V\setminus\{w\}\to\{0,1\}\\ \pi(\alpha)=1}} (f_\pi(w, 0) + f_\pi(w, 1)) \prod_{v\in V\setminus\{w\}} f_\pi(v, \pi(v))\\
&= \sum_{\substack{\pi:V\setminus\{w\}\to\{0,1\}\\ \pi(\alpha)=1}} \prod_{v\in V\setminus\{w\}} f_\pi(v, \pi(v)).
\end{aligned}$$

Note that in the penultimate line the sum goes over all assignments on $V \setminus \{w\}$ instead of V – the extraction of the $f_\pi(w, \cdot)$ term only works because w does not appear in any of the other factors. The last line uses the fact that the probabilities of complementary events sum up to 1.

The process yields another causal graph with one of the vertices removed; re-applying the argument inductively concludes the proof. □

Proposition A.2. *Algorithm 1 is correct with probability δ and terminates within time*

$$\mathcal{O}\left(n \cdot m \cdot \ln\left(\frac{2n}{\delta}\right) \cdot \varepsilon^{-3}\right).$$

Moreover, each computed value lies within an interval of $\pm\varepsilon$ around the true value.

Proof. Fix a root node $v_r \in V_R$. For $v \in V$ and $1 \leq i \leq N$, let $X_i(v)$ be the indicator variable that there is a path from v_r to v in the i-th random experiment. Observe that

$$\frac{1}{N}\sum_{i=1}^{N} \mathbb{E}[X_i(v)] = \mathbb{E}[X_0(v)] = \mathbb{P}[X_0(v) = 1],$$

that is, the quantity approximated by the algorithm (left hand-side) equals the probability that node v is reachable by v_r. So if the random experiments do not deviate too much from their expectations, the algorithm output is correct up to a certain relative error, which is determined in the following.

Define γ and N as in the algorithm. Note that $\gamma < \varepsilon < 1$.

Fix $v \in V$. Suppose for now that $\mu(v) \geq \gamma$. Using a two-sided Chernoff bound [164],

$$\begin{aligned}
&\mathbb{P}\left[\left|\sum_{i=1}^{N} X_i(v) - N\mu(v)\right| > N\mu(v)\varepsilon\right] \\
&\leq 2\exp\left(-\frac{\varepsilon^2}{3}N\gamma\right) \\
&= \frac{\delta}{n^2}. && \text{(by the choice of } N\text{)}
\end{aligned}$$

If however $\mu(v) \leq \gamma < \varepsilon$, using a one-sided Chernoff bound,

$$\begin{aligned}
&\mathbb{P}\left[\sum_{i=1}^{N} X_i(v)/N > \varepsilon\right] \\
&= \mathbb{P}\left[\sum_{i=1}^{N} X_i(v) > \left(1 + \frac{\varepsilon - \mu(v)}{\mu(v)}\right) N\mu(v)\right] \\
&\leq 2\exp\left(-\left(\frac{\varepsilon - \mu(v)}{\mu(v)}\right)^2 \frac{N\mu(v)}{3}\right) \\
&\leq 2\exp\left(-(\varepsilon - \gamma)^2 \frac{N}{3\gamma}\right) \\
&= 2\exp\left(-\left(\frac{\varepsilon - \gamma}{\varepsilon\gamma}\right)^2 \ln(2n^2/\delta)\right). && (*)
\end{aligned}$$

By the definition of γ it holds that $\varepsilon - \gamma > \varepsilon\gamma$ and thus $(*) \leq \frac{\delta}{n^2}$. To summarize, with probability at least δn^{-2} the following two statements hold:

- If $\mu(v) \geq \gamma$ then the relative error of the random experiment is at most ε; however, since $\mu(v) < 1$, this also implies that the absolute error $e(v) := \left|\sum_{i=1}^{N} X_i(v) - N\mu(v)\right|$ is at most ε.
- If $\mu(v) \leq \gamma$ then the absolute error $e(v)$ error is at most ε.

Note that the statements above hold for any fixed vertex v_0 and any fixed root node $v_{r,0}$. Using a union bound,

$$\mathbb{P}[\exists v_r \exists v : e(v) > \varepsilon] \leq n^2 \cdot \mathbb{P}[e(v_0) > \varepsilon] = \delta$$

yielding the desired error probability for the algorithm.

Regarding the running time, note that the inner FOR-loop can be implemented (e.g. using a breadth-first search) in linear time $\mathcal{O}(m)$ for each of the at most n root nodes, whereas the sampling requires time $\mathcal{O}(m)$, resulting in a total execution time of $\mathcal{O}(n \cdot m \cdot N)$. $\square$

Appendix B

Data sets

TABLE B.1 – Results of the simulation experiments

(varying size of input graph)

$\|V\|$	$\|E\|$	ε	δ	Iterations	Time (s)
100	515	0.1	0.01	57290	1.1163832
200	995	0.1	0.01	62764	1.8782055
300	1558	0.1	0.01	65966	3.4224201
400	2010	0.1	0.01	68238	4.0489518
500	2571	0.1	0.01	70000	5.2410411
600	2998	0.1	0.01	71440	6.290252
700	3463	0.1	0.01	72657	7.2129071
800	4079	0.1	0.01	73712	8.7000471
900	4499	0.1	0.01	74642	10.0516358
1000	5058	0.1	0.01	75474	11.1025155

(varying precision of algorithm output)

$\|V\|$	$\|E\|$			Iterations	Time (s)
500	2523	0.5	0.01	726	0.0577856
500	2392	0.2	0.01	9620	0.72131
500	2541	0.1	0.01	70000	5.1170565
500	2464	0.05	0.01	520596	31.2303292
500	2557	0.02	0.01	7587969	274.7254227

(varying algorithm error probability)

$\|V\|$	$\|E\|$			Iterations	Time (s)
500	2574	0.1	0.0001	88184	6.5013409
500	2521	0.1	0.001	79092	5.8744299
500	2479	0.1	0.01	70000	5.3008785
500	2540	0.1	0.1	60908	4.4340709

(varying average number of node neighbours)

$\|V\|$	$\|E\|$			Iterations	Time (s)
100	508	0.1	0.01	57290	0.74860860
200	980	0.1	0.01	62764	1.74781950
200	2004	0.1	0.01	62764	2.57128770
200	3957	0.1	0.01	62764	3.36202100
200	10067	0.1	0.01	62764	5.74008430
200	19992	0.1	0.01	62764	9.95537590
300	1491	0.1	0.01	65966	3.10028020
300	2956	0.1	0.01	65966	3.62702600
300	6032	0.1	0.01	65966	5.28465980

300	15068	0.1	0.01	65966	8.88317620
300	30075	0.1	0.01	65966	15.73896320
400	2002	0.1	0.01	68238	4.42690590
400	4007	0.1	0.01	68238	5.58108290
400	7993	0.1	0.01	68238	7.18062480
400	20094	0.1	0.01	68238	12.31568510
400	40110	0.1	0.01	68238	21.25638300
500	2497	0.1	0.01	70000	4.99750850
500	5091	0.1	0.01	70000	6.88772690
500	10048	0.1	0.01	70000	9.05478420
500	24883	0.1	0.01	70000	15.50143540
500	49828	0.1	0.01	70000	26.78152520
600	3012	0.1	0.01	71440	6.38884460
600	5897	0.1	0.01	71440	8.35841490
600	12163	0.1	0.01	71440	11.27393760
600	29795	0.1	0.01	71440	19.07167170
600	60180	0.1	0.01	71440	33.08584440
700	3393	0.1	0.01	72657	5.75831260
700	7078	0.1	0.01	72657	9.93901220
700	13910	0.1	0.01	72657	13.05916540
700	34940	0.1	0.01	72657	22.79221370
700	70164	0.1	0.01	72657	38.90024520
800	3936	0.1	0.01	73712	8.32702800
800	7963	0.1	0.01	73712	11.40233290
800	16012	0.1	0.01	73712	15.34878480
800	40093	0.1	0.01	73712	27.16270650
800	79989	0.1	0.01	73712	45.30316310
900	4426	0.1	0.01	74642	7.83409540
900	9219	0.1	0.01	74642	13.27833110
900	17995	0.1	0.01	74642	17.58192730
900	45347	0.1	0.01	74642	30.87114070
900	89958	0.1	0.01	74642	53.56402540
1000	5073	0.1	0.01	75474	11.19835840
1000	9867	0.1	0.01	75474	13.10751250
1000	20336	0.1	0.01	75474	19.48352650
1000	50238	0.1	0.01	75474	33.94672970
1000	99469	0.1	0.01	75474	58.11831390

List of Figures

List of Tables

Bibliography

[1] K. Stouffer, J. Falco and K. Scarfone, 'Guide to industrial control systems (ics) security', *NIST special publication*, vol. 800, no. 82, pp. 16–16, 2011.

[2] R. J. Turk, 'Cyber incidents involving control systems', Idaho National Laboratory (INL), Tech. Rep., 2005.

[3] M. A. McQueen, T. A. McQueen, W. F. Boyer and M. R. Chaffin, 'Empirical estimates and observations of 0day vulnerabilities', in *System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference on*, IEEE, 2009, pp. 1–12.

[4] S. L. Laboratory. (3rd May 2012). Qualitative vs. quantitative risk assessment, [Online]. Available: http://www.sans.edu/research/leadership-laboratory/article/risk-assessment (visited on 11/01/2015).

[5] L. A. T. Cox, D. Babayev and W. Huber, 'Some limitations of qualitative risk rating systems', *Risk Analysis*, vol. 25, no. 3, pp. 651–662, 2005.

[6] P. R. Garvey and Z. F. Lansdowne, 'Risk matrix: an approach for identifying, assessing, and ranking program risks', *Air Force Journal of Logistics*, vol. 22, no. 1, pp. 18–21, 1998.

[7] M. Willhite, 'Establishing a program risk baseline, an annotated briefing', *The MITRE Corporation, Bedford*, pp. 1–10, 1998.

[8] Q. Zhu, X. Kuang and Y. Shen, 'Risk matrix method and its application in the field of technical project risk management', *Engineering Science*, vol. 5, no. 1, pp. 89–94, 2003.

[9] H. Ni, A. Chen and N. Chen, 'Some extensions on risk matrix approach', *Safety Science*, vol. 48, no. 10, pp. 1269–1278, 2010.

[10] L. Anthony Tony Cox, 'What's wrong with risk matrices?', *Risk analysis*, vol. 28, no. 2, pp. 497–512, 2008.

[11] International Organization for Standardization, *ISO/IEC 27005: information security risk management*, 2008.

[12] C. N. de l'Informatique et des Libertés (CNIL). (2018). Privacy impact assessment (pia) 1 : methodology, [Online]. Available: https://www.cnil.fr/en/cnil-publishes-update-its-pia-guides (visited on 22/03/2018).

[13] ——, (2018). Privacy impact assessment (pia) 2 : template, [Online]. Available: https://www.cnil.fr/en/cnil-publishes-update-its-pia-guides (visited on 22/03/2018).

[14] ——, (2018). Privacy impact assessment (pia) 3 : knowledge bases, [Online]. Available: https://www.cnil.fr/en/cnil-publishes-update-its-pia-guides (visited on 22/03/2018).

[15] M. S. Lund, B. Solhaug and K. Stølen, *Model-driven risk analysis: the CORAS approach*. Springer Science & Business Media, 2010.

[16] Z. Yazar, 'A qualitative risk analysis and management tool–cramm', *SANS InfoSec Reading Room White Paper*, 2002.

[17] A. nationale de la sécurité des systèmes d'information, *EBIOS: expression des besoins et identification des objectifs de sécurité*, 2010.

[18] B. für Sicherheit in der Informationstechnik, *BSI-standard 200-3: risikomanagement*, 2017.

[19] M. A. Amutio, J. Candau and J. Mañas, 'Magerit–version 3, methodology for information systems risk analysis and management, book i – the method', *Ministerio de administraciones públicas*, 2014.

[20] CLUSIF and CLUSIQ, *MEHARI: principes fondamentaux et spécifications fonctionnelles*, 2010.

[21] C. J. Alberts and A. Dorofee, *Managing information security risks: the OCTAVE approach*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[22] ISACA, *The Risk IT framework*, 2009.

[23] G. Antoniou, M.-C. Saravanou and V. Stavrou, 'An overview of risk assessment methods', 2014. [Online]. Available: https://www.infosec.aueb.gr/Publications/2014-Poster%20RA%20Overview.pdf (visited on 22/01/2018).

[24] D. Ionita, 'Current established risk assessment methodologies and tools', 2013.

[25] F. Macedo and M. M. Da Silva, 'Comparative study of information security risk assessment models', *Universidad Técnica de Lisboa, Lisboa*,

[26] International Organization for Standardization, *ISO/IEC 27019: information technology – security techniques – information security management guidelines based on ISO/IEC 27002 for process control systems specific to the energy utility industry*, 2013.

[27] European Union Agency for Network and Information Security, *Baseline security recommendations for iot*, 2017.

[28] P. Y. Lipscy, K. E. Kushida and T. Incerti, 'The fukushima disaster and japan's nuclear plant vulnerability in comparative perspective', *Environmental science & technology*, vol. 47, no. 12, pp. 6082–6088, 2013.

[29] International Organization for Standardization, *ISO/IEC 31000: risk management*, 2018.

[30] E. Luiijf, A. Nieuwenhuijs, M. Klaver, M. van Eeten and E. Cruz, 'Empirical findings on critical infrastructure dependencies in europe', in *International Workshop on Critical Information Infrastructures Security*, Springer, 2008, pp. 302–310.

[31] S. M. Rinaldi, J. P. Peerenboom and T. K. Kelly, 'Identifying, understanding, and analyzing critical infrastructure interdependencies', *IEEE Control Systems*, vol. 21, no. 6, pp. 11–25, 2001.

[32] R. G. Little, 'Controlling cascading failure: understanding the vulnerabilities of interconnected infrastructures', *Journal of Urban Technology*, vol. 9, no. 1, pp. 109–123, 2002.

[33] S. McGee, J. Frittmann, S. Ahn and S. Murray, 'Risk relationship and cascading effects in critical infrastructures: implications for the hyogo framework (input paper prepared for the 2015 global assessment report on disaster risk reduction)', *Geneva, Switzerland, UNISDR*, 2014.

[34] S. Axelsson, 'The base-rate fallacy and the difficulty of intrusion detection', *ACM Transactions on Information and System Security (TISSEC)*, vol. 3, no. 3, pp. 186–205, 2000.

[35] T. Peng, C. Leckie and K. Ramamohanarao, 'Proactively detecting distributed denial of service attacks using source ip address monitoring', in *International Conference on Research in Networking*, Springer, 2004, pp. 771–782.

[36] D. Moore, C. Shannon, D. J. Brown, G. M. Voelker and S. Savage, 'Inferring internet denial-of-service activity', *ACM Transactions on Computer Systems (TOCS)*, vol. 24, no. 2, pp. 115–139, 2006.

[37] R. K. Chang, 'Defending against flooding-based distributed denial-of-service attacks: a tutorial', *IEEE communications magazine*, vol. 40, no. 10, pp. 42–51, 2002.

[38] A. Milenkoski, M. Vieira, S. Kounev, A. Avritzer and B. D. Payne, 'Evaluating computer intrusion detection systems: a survey of common practices', *ACM Computing Surveys (CSUR)*, vol. 48, no. 1, p. 12, 2015.

[39] V. Chandola, A. Banerjee and V. Kumar, 'Anomaly detection: a survey', *ACM computing surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.

[40] M. Botha, R. Von Solms, K. Perry, E. Loubser and G. Yamoyany, 'The utilization of artificial intelligence in a hybrid intrusion detection system', in *Proceedings of the 2002 annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology*, South African Institute for Computer Scientists and Information Technologists, 2002, pp. 149–155.

[41] C. Katar, 'Combining multiple techniques for intrusion detection', *Int J Comput Sci Network Security*, vol. 6, no. 2B, pp. 208–218, 2006.

[42] S. Axelsson, 'Intrusion detection systems: a survey and taxonomy', Technical report, Tech. Rep., 2000.

[43] H. Debar, M. Dacier and A. Wespi, 'Towards a taxonomy of intrusion-detection systems', *Computer Networks*, vol. 31, no. 8, pp. 805–822, 1999.

[44] R. Sommer and V. Paxson, 'Outside the closed world: on using machine learning for network intrusion detection', in *Security and Privacy (SP), 2010 IEEE Symposium on*, IEEE, 2010, pp. 305–316.

[45] M. Barreno, B. Nelson, R. Sears, A. D. Joseph and J. D. Tygar, 'Can machine learning be secure?', in *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, ACM, 2006, pp. 16–25.

[46] S. Muller, 'Risk monitoring in industrial control systems', in *Advanced Data Collection and Risks (ADaCoR) Workshop 2016*, 2016.

[47] B. Fetler and S. Muller, 'Dynamic risk analysis', in *Security Assessment for Systems, Services, and Infrastructures (SASSI) Workshop 2015*, 2015.

[48] S. Muller, C. Harpes, Y. Le Traon, S. Gombault, J.-M. Bonnin and P. Hoffmann, 'Dynamic risk analyses and dependency-aware root cause model for critical infrastructures', in *Critical Information Infrastructures Security: 11th International Conference, CRITIS 2016, Paris, France, October 10–12, 2016, Revised Selected Papers*, G. Havarneanu, R. Setola, H. Nassopoulos and S. Wolthusen, Eds. Cham: Springer

International Publishing, 2017, pp. 163–175, ISBN: 978-3-319-71368-7. DOI: 10.1007/978-3-319-71368-7_14. [Online]. Available: https://doi.org/10.1007/978-3-319-71368-7_14.

[49] O. Gadyatskaya, C. Harpes, S. Mauw, C. Muller and S. Muller, 'Bridging two worlds: reconciling practical risk assessment methodologies with theory of attack trees', in *Graphical Models for Security: Third International Workshop, GraMSec 2016, Lisbon, Portugal, June 27, 2016, Revised Selected Papers*, B. Kordy, M. Ekstedt and S. D. Kim, Eds. Springer International Publishing, 2016, pp. 80–93, ISBN: 978-3-319-46263-9. DOI: 10.1007/978-3-319-46263-9_5. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-46263-9_5.

[50] S. Muller, C. Harpes and C. Muller, 'Fast and optimal countermeasure selection for attack defence trees', in *Risk Assessment and Risk-Driven Quality Assurance: 4th International Workshop, RISK 2016, Held in Conjunction with ICTSS 2016, Graz, Austria, October 18, 2016, Revised Selected Papers*, J. Großmann, M. Felderer and F. Seehusen, Eds. Cham: Springer International Publishing, 2017, pp. 53–65, ISBN: 978-3-319-57858-3. DOI: 10.1007/978-3-319-57858-3_5. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-57858-3_5.

[51] S. Muller, C. Harpes, Y. L. Traon, S. Gombault and J.-M. Bonnin, 'Efficiently computing the likelihoods of cyclically interdependent risk scenarios', *Computers & Security*, vol. 64, pp. 59–68, 2017, ISSN: 0167-4048. DOI: http://dx.doi.org/10.1016/j.cose.2016.09.008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167404816301158.

[52] S. Muller, J. Lancrenon, C. Harpes, Y. L. Traon, S. Gombault and J.-M. Bonnin, 'A training-resistant anomaly detection system', *Computers & Security*, vol. 76, pp. 1–11, 2018, ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2018.02.015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016740481830155X.

[53] X. Tong and X. Ban, 'A hierarchical information system risk evaluation method based on asset dependence chain', *International Journal of Security and Its Applications*, vol. 8, no. 6, pp. 81–88, 2014.

[54] F. Den Braber, I. Hogganvik, M. Lund, K. Stølen and F. Vraalsen, 'Model-based security analysis in seven steps—a guided tour to the coras method', *BT Technology Journal*, vol. 25, no. 1, pp. 101–117, 2007.

[55] J. Breier, 'Asset valuation method for dependent entities', *Journal of Internet Services and Information Security (JISIS)*, vol. 4, no. 3, pp. 72–81, 2014.

[56] N. Liu, J. Zhang and X. Wu, 'Asset analysis of risk assessment for iec 61850-based power control systems—part i: methodology', *IEEE Transactions on Power Delivery*, vol. 26, no. 2, pp. 869–875, 2011.

[57] B. Suh and I. Han, 'The is risk analysis based on a business model', *Information & Management*, vol. 41, no. 2, pp. 149–158, 2003.

[58] 'Event tree analysis (eta)', *IEC60300-3-9, Dependability Management – Part 3: Application Guide – Section 9: Risk Analysis of Technological Systems, first ed.*, 1995.

[59] P. Giorgini, J. Mylopoulos, E. Nicchiarelli and R. Sebastiani, 'Formal reasoning techniques for goal models', *J. Data Semantics*, vol. 1, no. 1, pp. 1–20, 2003.

[60] E. Navarro, P. Letelier, D. Reolid and I. Ramos, 'Configurable satisfiability propagation for goal', *Advances in information systems development: new methods and practice for the networked society*, vol. 2, p. 167, 2007.

[61] B. Schneier, 'Attack trees', *Dr. Dobb's journal*, vol. 24, no. 12, pp. 21–29, 1999.

[62] S. Mauw and M. Oostdijk, 'Foundations of attack trees', in *Icisc*, Springer, vol. 3935, 2005, pp. 186–198.

[63] S. Evans and J. Wallner, 'Risk-based security engineering through the eyes of the adversary', in *Information Assurance Workshop, 2005. IAW'05. Proceedings from the Sixth Annual IEEE SMC*, IEEE, 2005, pp. 158–165.

[64] P. Schweitzer, 'Attack-defense trees.', PhD thesis, University of Twente, Enschede, Netherlands, 2013.

[65] B. Kordy, S. Mauw, S. Radomirović and P. Schweitzer, 'Attack–defense trees', *Journal of Logic and Computation*, vol. 24, no. 1, pp. 55–87, 2014.

[66] F. Baiardi and D. Sgandurra, 'Assessing ICT risk through a Monte Carlo method', *Environment Systems and Decisions*, vol. 33, no. 4, pp. 486–499, 2013.

[67] T. R. Ingoldsby, 'Attack tree-based threat risk analysis', *Amenaza Technologies Limited*, pp. 3–9, 2010.

[68] L. Grunske and D. Joyce, 'Quantitative risk-based security prediction for component-based systems with explicitly modeled attack profiles', *Journal of Systems and Software*, vol. 81, no. 8, pp. 1327–1345, 2008.

[69] K. S. Edge, G. C. Dalton, R. A. Raines and R. F. Mills, 'Using attack and protection trees to analyze threats and defenses to homeland security', in *Military Communications Conference, 2006. MILCOM 2006. IEEE*, IEEE, 2006, pp. 1–7.

[70] M. A. McQueen, W. F. Boyer, M. A. Flynn and G. A. Beitel, 'Quantitative cyber risk reduction estimation methodology for a small scada control system', in *System Sciences, 2006. HICSS'06. Proceedings of the 39th Annual Hawaii International Conference on*, IEEE, vol. 9, 2006, pp. 226–226.

[71] I. B. Utne, P. Hokstad and J. Vatn, 'A method for risk modeling of interdependencies in critical infrastructures', *Reliability Engineering & System Safety*, vol. 96, no. 6, pp. 671–678, 2011.

[72] T. W. Kwan and H. K. Leung, 'A risk management methodology for project risk dependencies', *IEEE Transactions on Software Engineering*, vol. 37, no. 5, pp. 635–648, 2011.

[73] R. Dantu, K. Loper and P. Kolan, 'Risk management using behavior based attack graphs', in *Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004. International Conference on*, IEEE, vol. 1, 2004, pp. 445–449.

[74] S. Fenz, M. Hudec *et al.*, 'Ontology-based generation of Bayesian networks', in *Complex, Intelligent and Software Intensive Systems, 2009. CISIS'09. International Conference on*, IEEE, 2009, pp. 712–717.

[75] B. Rahmad, S. H. Supangkat, J. Sembiring and K. Surendro, 'Modeling asset dependency for security risk analysis using threat-scenario dependency', *International Journal of Computer Science and Information Security*, vol. 10, no. 4, p. 103, 2012.

[76] N. Poolsappasit, R. Dewri and I. Ray, 'Dynamic security risk management using bayesian attack graphs', *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 1, pp. 61–74, 2012.

[77] J. Homer, X. Ou and D. Schmidt, 'A sound and practical approach to quantifying security risk in enterprise networks', *Kansas State University Technical Report*, pp. 1–15, 2009.

[78] P. Kotzanikolaou, M. Theoharidou and D. Gritzalis, 'Assessing n-order dependencies between critical infrastructures', *International Journal of Critical Infrastructures 6*, vol. 9, no. 1-2, pp. 93–110, 2013.

[79] L. Wang, T. Islam, T. Long, A. Singhal and S. Jajodia, 'An attack graph-based probabilistic security metric', in *Data and applications security XXII*, Springer, 2008, pp. 283–296.

[80] A. Årnes, K. Sallhammar, K. Haslum, T. Brekne, M. E. G. Moe and S. J. Knapskog, 'Real-time risk assessment with network sensors and intrusion detection systems', in *International Conference on Computational and Information Science*, Springer, 2005, pp. 388–397.

[81] K. Haslum and A. Arnes, 'Multisensor real-time risk assessment using continuous-time hidden markov models', in *Computational Intelligence and Security, 2006 International Conference on*, IEEE, vol. 2, 2006, pp. 1536–1540.

[82] X. Tan, Y. Zhang, X. Cui and H. Xi, 'Using hidden markov models to evaluate the real-time risks of network', in *Knowledge Acquisition and Modeling Workshop, 2008. KAM Workshop 2008. IEEE International Symposium on*, IEEE, 2008, pp. 490–493.

[83] W. Kanoun, S. Dubus, S. Papillon, N. Cuppens-Boulahia and F. Cuppens, 'Towards dynamic risk management: success likelihood of ongoing attacks', *Bell Labs Technical Journal*, vol. 17, no. 3, pp. 61–78, 2012.

[84] M. Jahnke, C. Thul and P. Martini, 'Graph based metrics for intrusion response measures in computer networks', in *Local Computer Networks, 2007. LCN 2007. 32nd IEEE Conference on*, IEEE, 2007, pp. 1035–1042.

[85] S. Noel, S. Jajodia, L. Wang and A. Singhal, 'Measuring security risk of networks using attack graphs', *International Journal of Next-Generation Computing*, vol. 1, no. 1, pp. 135–147, 2010.

[86] P. Xie, J. H. Li, X. Ou, P. Liu and R. Levy, 'Using bayesian networks for cyber security analysis', in *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP international conference on*, IEEE, 2010, pp. 211–220.

[87] J. Homer and X. Ou, 'Sat-solving approaches to context-aware enterprise network security management', *IEEE Journal on selected areas in communications*, vol. 27, no. 3, 2009.

[88] A. Gehani and G. Kedem, 'Rheostat: real-time risk management', in *RAID*, Springer, 2004, pp. 296–314.

[89] M. Paté-Cornell, P. J. Regan *et al.*, 'Dynamic risk management systems: hybrid architecture and offshore platform illustration', *Risk analysis*, vol. 18, no. 4, pp. 485–496, 1998.

[90] W. Kanoun, N. Cuppens-Boulahia, F. Cuppens and F. Autrel, 'Advanced reaction using risk assessment in intrusion detection systems', in *International Workshop on Critical Information Infrastructures Security*, Springer, 2007, pp. 58–70.

[91] M. Giannakis and M. Louis, 'A multi-agent based framework for supply chain risk management', *Journal of Purchasing and Supply Management*, vol. 17, no. 1, pp. 23–31, 2011.

[92] J. Jiang, P. Wang, W.-s. Lung, L. Guo and M. Li, 'A gis-based generic real-time risk assessment framework and decision tools for chemical spills in the river basin', *Journal of hazardous materials*, vol. 227, pp. 280–291, 2012.

[93] N. Dulac, 'A framework for dynamic safety and risk management modeling in complex engineering systems', PhD thesis, Massachusetts Institute of Technology, 2007.

[94] J. P. Anderson *et al.*, 'Computer security threat monitoring and surveillance', James P. Anderson Company, Fort Washington, Pennsylvania, Tech. Rep., 1980.

[95] D. E. Denning, 'An intrusion-detection model', *IEEE Transactions on software engineering*, no. 2, pp. 222–232, 1987.

[96] R. Mitchell and I.-R. Chen, 'A survey of intrusion detection techniques for cyber-physical systems', *ACM Computing Surveys (CSUR)*, vol. 46, no. 4, p. 55, 2014.

[97] B. Zhu and S. Sastry, 'Scada-specific intrusion detection/prevention systems: a survey and taxonomy', in *Proceedings of the 1st Workshop on Secure Control Systems (SCS)*, vol. 11, 2010.

[98] A. L. Buczak and E. Guven, 'A survey of data mining and machine learning methods for cyber security intrusion detection', *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1153–1176, 2016.

[99] H. Brahmi, I. Brahmi and S. B. Yahia, 'Omc-ids: at the cross-roads of olap mining and intrusion detection', in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Springer, 2012, pp. 13–24.

[100] L. Portnoy, E. Eskin and S. Stolfo, 'Intrusion detection with unlabeled data using clustering', in *In Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001*, Citeseer, 2001.

[101] Z. Muda, W. Yassin, M. Sulaiman and N. Udzir, 'Intrusion detection based on k-means clustering and naive bayes classification', in *Information Technology in Asia (CITA 11), 2011 7th International Conference on*, IEEE, 2011, pp. 1–6.

[102] V. Barot and D. Toshniwal, 'A new data mining based hybrid network intrusion detection model', in *Data Science & Engineering (ICDSE), 2012 International Conference on*, IEEE, 2012, pp. 52–57.

[103] K. Sequeira and M. Zaki, 'Admit: anomaly-based data mining for intrusions', in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2002, pp. 386–395.

[104] A. Almalawi, 'Designing unsupervised intrusion detection for scada systems', 2014.

[105] R. M. Elbasiony, E. A. Sallam, T. E. Eltobely and M. M. Fahmy, 'A hybrid network intrusion detection framework based on random forests and weighted k-means', *Ain Shams Engineering Journal*, vol. 4, no. 4, pp. 753–762, 2013.

[106] L. Tomlin, M. Farnam and S. Pan, 'A clustering approach to industrial network intrusion detection', in *Proceedings of the 2016 Information Security Research and Education (INSuRE) Conference (INSuRECon-16)*, 2016.

[107] W. Wang, T. Guyet, R. Quiniou, M.-O. Cordier, F. Masseglia and X. Zhang, 'Autonomic intrusion detection: adaptively detecting anomalies over unlabeled audit data streams in computer networks', *Knowledge-Based Systems*, vol. 70, pp. 103–117, 2014.

[108] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, 'A density-based algorithm for discovering clusters in large spatial databases with noise.', in *Kdd*, vol. 96, 1996, pp. 226–231.

[109] M. Blowers and J. Williams, 'Machine learning applied to cyber operations', in *Network Science and Cybersecurity*, Springer, 2014, pp. 155–175.

[110] S. Shamshirband, A. Amini, N. B. Anuar, M. L. M. Kiah, Y. W. Teh and S. Furnell, 'D-ficca: a density-based fuzzy imperialist competitive clustering algorithm for intrusion detection in wireless sensor networks', *Measurement*, vol. 55, pp. 212–226, 2014.

[111] A. Amini, H. Saboohi, T. Ying Wah and T. Herawan, 'A fast density-based clustering algorithm for real-time internet of things stream', *The Scientific World Journal*, vol. 2014, 2014.

[112] K. Leung and C. Leckie, 'Unsupervised anomaly detection in network intrusion detection using clusters', in *Proceedings of the Twenty-eighth Australasian conference on Computer Science-Volume 38*, Australian Computer Society, Inc., 2005, pp. 333–342.

[113] G. R. Hendry and S. J. Yang, 'Intrusion signature creation via clustering anomalies', in *Data Mining, Intrusion Detection, Information Assurance, and Data Networks Security 2008*, International Society for Optics and Photonics, vol. 6973, 2008, p. 69730C.

[114] X. Wang, C. Zhang and K. Zheng, 'Intrusion detection algorithm based on density, cluster centers, and nearest neighbors', *China Communications*, vol. 13, no. 7, pp. 24–31, 2016.

[115] C. Harpes, A. Adelsbach, S. Zatti and N. Peccia, 'Quantitative risk assessment with isamm on esa's operations data system', *Proceedings of TTC*, pp. 173–176, 2007.

[116] J. Pearl, *Causality: Models, Reasoning, and Inference*. New York, NY, USA: Cambridge University Press, 2000, ISBN: 0-521-77362-8.

[117] N. Idika and B. Bhargava, 'Extending attack graph-based security metrics and aggregating their application', *Dependable and Secure Computing, IEEE Transactions on*, vol. 9, no. 1, pp. 75–85, 2012.

[118] G. F. Cooper, 'The computational complexity of probabilistic inference using bayesian belief networks', *Artificial intelligence*, vol. 42, no. 2, pp. 393–405, 1990.

[119] N. L. Zhang and D. Poole, 'A simple approach to bayesian network computations', in *Proc. of the Tenth Canadian Conference on Artificial Intelligence*, 1994.

[120] H. Kiiveri, T. P. Speed and J. B. Carlin, 'Recursive causal models', *Journal of the Australian Mathematical Society (Series A)*, vol. 36, no. 01, pp. 30–52, 1984.

[121] S. Russell, P. Norvig and A. Intelligence, 'A modern approach', *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, vol. 25, p. 27, 1995.

[122] D. Grochocki, J. H. Huh, R. Berthier, R. Bobba, W. H. Sanders, A. A. Cárdenas and J. G. Jetcheva, 'AMI threats, intrusion detection requirements and deployment recommendations', in *Smart Grid Communications (SmartGridComm), 2012 IEEE Third International Conference on*, IEEE, 2012, pp. 395–400.

[123] ENISA. (2016). Communication network interdependencies in smart grids, [Online]. Available: https://www.enisa.europa.eu/publications/communication-network-interdependencies-in-smart-grids (visited on 28/04/2016).

[124] P. Clote and E. Kranakis, *Boolean functions and computation models*. Springer Science & Business Media, 2013.

[125] B. Kordy, S. Mauw, S. Radomirović and P. Schweitzer, 'Attack–defense trees', *Journal of Logic and Computation*, exs029, 2012.

[126] ——, 'Foundations of attack–defense trees', in *International Workshop on Formal Aspects in Security and Trust*, Springer, 2010, pp. 80–95.

[127] B. Kordy, S. Mauw, M. Melissen and P. Schweitzer, 'Attack–defense trees and two-player binary zero-sum extensive form games are equivalent', in *International Conference on Decision and Game Theory for Security*, Springer, 2010, pp. 245–256.

[128] International Organization for Standardization, *ISO/IEC 27002 – information technology – security techniques – code of practice for information security management*, 2013.

[129] D. G. Luenberger, *Introduction to linear and nonlinear programming*. Addison-Wesley Reading, MA, 1973, vol. 28.

[130] A. Roy, D. S. Kim and K. S. Trivedi, 'Scalable optimal countermeasure selection using implicit enumeration on attack countermeasure trees', in *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*, IEEE, 2012, pp. 1–12.

[131] E. W. Dijkstra, 'A note on two problems in connexion with graphs', *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[132] M. L. Fredman and R. E. Tarjan, 'Fibonacci heaps and their uses in improved network optimization algorithms', *Journal of the ACM (JACM)*, vol. 34, no. 3, pp. 596–615, 1987.

[133] R. W. Floyd, 'Algorithm 97: shortest path', *Communications of the ACM*, vol. 5, no. 6, p. 345, 1962.

[134] (1999). KDD Cup, [Online]. Available: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html (visited on 01/06/2017).

[135] E. Cole, *Advanced persistent threat: understanding the danger and how to protect your organization*. Newnes, 2012.

[136] S. Garcia, M. Grill, J. Stiborek and A. Zunino, 'An empirical comparison of botnet detection methods', *computers & security*, vol. 45, pp. 100–123, 2014.

[137] R. Berthier, D. I. Urbina, A. A. Cárdenas, M. Guerrero, U. Herberg, J. G. Jetcheva, D. Mashima, J. H. Huh and R. B. Bobba, 'On the practicality of detecting anomalies with encrypted traffic in AMI', in *Smart Grid Communications (SmartGridComm), 2014 IEEE International Conference on*, IEEE, 2014, pp. 890–895.

[138] D. Wagner and P. Soto, 'Mimicry attacks on host-based intrusion detection systems', in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, ACM, 2002, pp. 255–264.

[139] D. Corpora. (2009). M57 patents, [Online]. Available: http://digitalcorpora.org/corpora/scenarios/m57-patents-scenario (visited on 17/07/2017).
[140] J. A. Silva, E. R. Faria, R. C. Barros, E. R. Hruschka, A. C. de Carvalho and J. Gama, 'Data stream clustering: a survey', *ACM Computing Surveys (CSUR)*, vol. 46, no. 1, p. 13, 2013.
[141] Y. Chen and L. Tu, 'Density-based clustering for real-time stream data', in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2007, pp. 133–142.
[142] K. Udommanetanakit, T. Rakthanmanon and K. Waiyamai, 'E-stream: evolution-based technique for stream clustering', *Advanced Data Mining and Applications*, pp. 605–615, 2007.
[143] D. Corpora. (2008). Nitroba university harassment scenario, [Online]. Available: http://digitalcorpora.org/corpora/scenarios/nitroba-university-harassment-scenario (visited on 26/07/2017).
[144] N. AB. (2015). 4SICS geek lounge SCADA network capture, [Online]. Available: http://www.netresec.com/?page=PCAP4SICS (visited on 26/07/2017).
[145] S. Mirzaie, A. K. Elyato and M. A. Sarram, 'Preventing of syn flood attack with iptables firewall', in *Communication Software and Networks, 2010. ICCSN'10. Second International Conference on*, IEEE, 2010, pp. 532–535.
[146] M. Ihde and W. H. Sanders, 'Barbarians in the gate: an experimental validation of nic-based distributed firewall performance and flood tolerance', in *Dependable Systems and Networks, 2006. DSN 2006. International Conference on*, IEEE, 2006, pp. 209–216.
[147] J. Jung, V. Paxson, A. W. Berger and H. Balakrishnan, 'Fast portscan detection using sequential hypothesis testing', in *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on*, IEEE, 2004, pp. 211–225.
[148] A. Sridharan, T. Ye and S. Bhattacharyya, 'Connectionless port scan detection on the backbone', in *Performance, Computing, and Communications Conference, 2006. IPCCC 2006. 25th IEEE International*, IEEE, 2006, 10–pp.
[149] C. B. Lee, C. Roedel and E. Silenok, 'Detection and characterization of port scan attacks', *Univeristy of California, Department of Computer Science and Engineering*, 2003.
[150] L. Aniello, G. Lodi and R. Baldoni, 'Inter-domain stealthy port scan detection through complex event processing', in *Proceedings of the 13th European Workshop on Dependable Computing*, ACM, 2011, pp. 67–72.
[151] S. C. Lee and D. V. Heinbuch, 'Training a neural-network based intrusion detector to recognize novel attacks', *IEEE Transactions on systems, man, and Cybernetics-Part A: Systems and Humans*, vol. 31, no. 4, pp. 294–299, 2001.
[152] A. Khalimonenko, O. Kupreev and K. Ilganaev, 'Kaspersky ddos intelligence report for q4 2017', Tech. Rep., 2018. [Online]. Available: https://securelist.com/ddos-attacks-in-q4-2017/83729/ (visited on 04/04/2018).

[153] D. Holmes, 'The ddos threat spectrum', Tech. Rep., 2012. [Online]. Available: https://f5.com/resources/white-papers/the-ddos-threat-spectrum (visited on 04/04/2018).

[154] J. Jeong, H. Kim and J. Park, 'Requirements for security services based on software-defined networking', *IETF*, 2014.

[155] S. Jin and D. S. Yeung, 'A covariance analysis model for ddos attack detection', in *Communications, 2004 IEEE International Conference on*, IEEE, vol. 4, 2004, pp. 1882–1886.

[156] S. Yu, W. Zhou and R. Doss, 'Information theory based detection against network behavior mimicking ddos attacks', *IEEE Communications Letters*, vol. 12, no. 4, 2008.

[157] V. Sekar, N. G. Duffield, O. Spatscheck, J. E. van der Merwe and H. Zhang, 'Lads: large-scale automated ddos detection system.', in *USENIX Annual Technical Conference, General Track*, 2006, pp. 171–184.

[158] A. Lakhina, M. Crovella and C. Diot, 'Mining anomalies using traffic feature distributions', in *ACM SIGCOMM Computer Communication Review*, ACM, vol. 35, 2005, pp. 217–228.

[159] W. Press, S. Teukolsky, W. Vetterling and B. Flannery, *Numerical Recipes in Fortran 77: The Art of Scientific Computing. Cambridge Univ.* Press, Cambridge, 1992.

[160] B. Brykczynski and R. A. Small, 'Reducing internet-based intrusions: effective security patch management', *IEEE software*, vol. 20, no. 1, pp. 50–57, 2003.

[161] Microsoft, 'Microsoft security intelligence report', Tech. Rep., 2013.

[162] I. FIRST.org. (2015). Common vulnerability scoring system v3.0: specification document, [Online]. Available: https://www.first.org/cvss/specification-document (visited on 29/03/2018).

[163] I. Arce, 'The weakest link revisited [information security]', *IEEE Security & Privacy*, vol. 99, no. 2, pp. 72–76, 2003.

[164] R. Motwani and P. Raghavan, *Randomized algorithms*. Chapman & Hall/CRC, 2010.

Liste des publications de Steve Muller

- S. Muller, C. Harpes, Y. Le Traon, S. Gombault, J.-M. Bonnin and P. Hoffmann, 'Dynamic risk analyses and dependency-aware root cause model for critical infrastructures', in Critical Information Infrastructures Security: 11th International Conference, CRITIS 2016, Paris, France, October 10–12, 2016, Revised Selected Papers, G. Havarneanu, R. Setola, H. Nassopoulos and S.Wolthusen, Eds. Cham: Springer International Publishing, 2017, pp. 163–175, ISBN: 978-3-319-71368-7. DOI: 10.1007/978-3-319-71368-7_14.
- O. Gadyatskaya, C. Harpes, S. Mauw, C. Muller and S. Muller, 'Bridging two worlds: Reconciling practical risk assessment methodologies with theory of attack trees', in Graphical Models for Security: Third International Workshop, GraMSec 2016, Lisbon, Portugal, June 27, 2016, Revised Selected Papers, B. Kordy, M. Ekstedt and S. D. Kim, Eds. Springer International Publishing, 2016, pp. 80–93, ISBN: 978-3-319-46263-9. DOI: 10.1007/978-3-319-46263-9_5.
- S. Muller, C. Harpes and C. Muller, 'Fast and optimal countermeasure selection for attack defence trees', in Risk Assessment and Risk-Driven Quality Assurance: 4th International Workshop, RISK 2016, Held in Conjunction with ICTSS 2016, Graz, Austria, October 18, 2016, Revised Selected Papers, J. Großmann, M. Felderer and F. Seehusen, Eds. Cham: Springer International Publishing, 2017, pp. 53–65, ISBN: 978-3-319-57858-3. DOI: 10.1007/978-3-319-57858-3_5.
- S. Muller, C. Harpes, Y. L. Traon, S. Gombault and J.-M. Bonnin, 'Efficiently computing the likelihoods of cyclically interdependent risk scenarios', Computers & Security, vol. 64, pp. 59–68, 2017, ISSN: 0167-4048. DOI: http://dx.doi.org/10.1016/j.cose.2016.09.008.
- Steve Muller, Jean Lancrenon, Carlo Harpes, Yves Le Traon, Sylvain Gombault, Jean-Marie Bonnin, 'A training-resistant anomaly detection system', Computers & Security, vol. 76, pp. 1–11, 2018, ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2018.02.015.